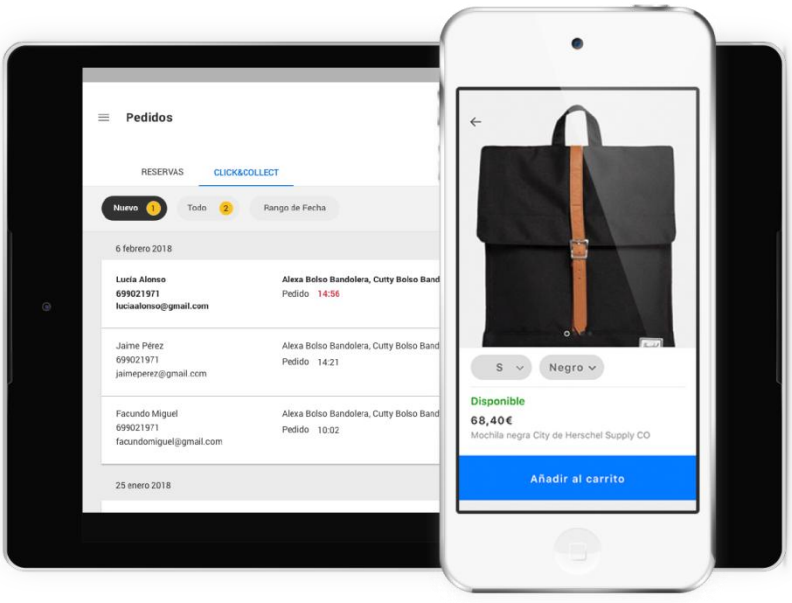
UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Desarrollo de una app para mejorar la experiencia de compra al por menor

Memoria final



Autor: Adrián Ruiz Cordón
Director: Luis Manuel Pérez Serna
Ponente: María José Casany
Departamento: Ingeniería del Software
25 Enero de 2019

Abstract

En la actualidad, la manera en la que se compra ha cambiado mucho respecto a las prácticas de hace 20 o 30 años. Se ha pasado de ir a los comercios de barrio de toda la vida a comprar en grandes superficies. Este cambio ha ido en paralelo con un gran avance tecnológico, el cual se ve reflejado en la creciente popularidad de las tiendas online. Aún así, el comercio físico en ocasiones no ha sabido adaptarse a los nuevos tiempos y cada vez menos gente va a comprar presencialmente. En esos comercios, los clientes se encuentran con una experiencia de compra desfasada, con colas largas y con falta de atención personalizada. Para solucionar todo esto, desde la *start-up BuyYourself* se está implementado un sistema para modernizar como se vende y se compra al por menor en tiendas físicas. Para ello, se está llevando a cabo el desarrollo de varios módulos y aplicaciones. Entre ellas está el sujeto de este proyecto: *ASApp (Assistant Shop App)*, una aplicación para dispositivos *iPhone* desde la cual los dependientes de la tienda pueden consultar productos, cerrar ventas y gestionar pedidos y reservas. De esta manera, se puede ofrecer un trato más personalizado a los clientes y se les puede atender desde cualquier punto de la tienda, reduciendo así los tiempos de compra.

En la actualitat, la manera en la que es compra ha canviat molt respecte a les pràctiques de fa 20 o 30 anys. S'ha passat d'anar als comerços de barri de tota la vida a comprar a grans superfícies. Aquest canvi ha anat en paral·lel amb un gran avenç tecnològic. Aquest avenç es veu reflectit en la popularitat de les botigues online. No obstant, el comerç físic en ocasions no ha sapigut adaptar-se als nous temps i cada vegada menys gent va a comprar presencialment. Els clients es troben amb una experiència de compra desfasada, amb cues llargues i amb una falta d'atenció personalitzada. Per solucionar tot això, des de la *start-up BuyYourself* s'està implementant un sistema per modernitzar com es ven i es compra al detall en botigues físiques. Per aconseguir-ho, s'està duent a terme el desenvolupament de diversos mòduls i aplicacions. Entre elles està el subjecte principal d'aquest projecte: *ASApp (Assistant Shop App)*, una aplicació per dispositius *iPhone* des de la que els dependents de la botiga puguin consultar productes, tancar vendes i gestionar comandes i reserves. D'aquesta manera, es pot oferir un tracte més personalitzat als clients i se'ls pot atendre des de qualsevol punt de la botiga reduint així els temps de compra.

Nowadays, the way in which we buy has changed a lot compared to how it was 20 or 30 years ago. We have gone from buying from the local stores to doing it in the shops of big companies. This change has been parallel to a big technologic innovation, something that is reflected in the increasing popularity of the online shops. Traditional shops, in occasions, have not been able to adapt themselves to the new tendencies of shopping, and less and less people buy from them as time passes. In those shops, the clients find themselves with an old-fashioned way of acquiring products, long queues and a lack of customized attention. To bring a solution to this, from the *start-up BuyYourself* a new system to modernize the way we sell and buy in the physical stores has been implemented. To do so, different modules and applications are being developed. The subject of this project is among them: *ASApp (Assisttant Shop App)*, an *iPhone* application from which the shop assistants can consult products, close sales and control orders and reservations. Thus, it can offer a more customized treatment to the customers and it can attend them from any place of the store, reducing then the purchase time.

Glosario de conceptos básicos

start-up: empresa que busca arrancar, emprender o montar un nuevo negocio, y alude a ideas de negocios que están empezando o están en construcción. Generalmente se trata de empresas emergentes apoyadas en la tecnología.

retail: venta al por menor.

retailers: empresas o negocios de venta al por menor.

eCommerce: distribución, venta, compra, marketing y suministro de información de productos o servicios a través de Internet.

self-checkout: servicio de autopago que proporciona un mecanismo para que los clientes procesen sus propias compras en un minorista a través de un dispositivo.

omnicanalidad: es la integración de todos los canales existentes en el mercado, para tal de generar caminos que se interrelacionen. Esto implica que un cliente que inició una comunicación por una vía de interacción pueda continuarla por otra.

front-end: es la parte del *software* que interactúa con los usuarios y el back-end**.**

back-end: es la parte del *software* que se toma los datos, los procesa y los envía al usuario; además de encargarse de las consultas o peticiones a la Base de Datos, la conexión con el servidor.

sprint: intervalo de tiempo prefijado en el que se dividen las fases de un proyecto.

Índice de contenidos

Índice de Tablas y Figuras

1. Contextualización

1.1. Introducción

En los últimos tiempos, el comercio al por menor ha sufrido importantes cambios. Antes todas las compras se realizaban en pequeños comercios familiares, y ahora cada vez más, nos encontramos con que las grandes franquicias se abren paso en las calles de pueblos y ciudades. Estos cambios han sido contemporáneos al constante avance tecnológico al que hemos llegado. Las nuevas tecnologías juegan un papel muy importante en la sociedad actual, y también en la manera de comprar. El comercio online es uno de los métodos más usados actualmente.

No solo las tecnologías han cambiado la experiencia de compra en por menor. La sociedad ha llegado a un punto en el que cada segundo cuenta. Cada vez queremos tener las cosas antes, ahorrándonos tiempo y esfuerzo. Y uno de los factores de los que más se suelen quejar los consumidores es de las largas colas en las tiendas.

Todo esto ha revolucionado la manera de comprar productos. La mayoría de comercios ofrecen servicios como asistencia y gestión de incidencias por internet, compra y encargo de productos a través de la red o aplicaciones para poder utilizar todos sus servicios. Incluso empresas como *Amazon*[1] han llegado a crear una forma de poder entrar en una tienda, coger los productos y salir de ella sin tener cajas: *Amazon Go*[1]. Con una aplicación y registrando virtualmente los productos que coge el usuario, es capaz de poder registrar la compra al salir, cobrarla y evitar colas.

La gran mayoría de comercios y franquicias han empezado ya su camino para revolucionar la manera en la que venden y gestionan sus negocios para poder ser más eficientes y obtener más beneficios. Varias empresas del sector tecnológico han visto un filón en este mercado y han desarrollado algunas soluciones, pero la mayoría son muy específicas para un tipo de comercios o un aspecto de estos en concreto. Es por eso que desde la *start-up* *BuyYourself* hemos decidido aportar una solución que mejore la experiencia de compra al por menor tanto para consumidores como para vendedores utilizando tecnologías avanzadas.

En concreto, el sujeto de este proyecto es el desarrollo de la aplicación para vendedores en dispositivos *iPhone*. Esta aplicación permitirá que los dependientes puedan otorgar una atención más completa y personalizada a sus clientes, ya que podrán desarrollar todas sus funciones rápidamente desde la aplicación y así poder dedicarles más tiempo. Así mismo, esta aplicación ayudará a tener constancia de los pedidos que se hacen para recoger en la tienda para gestionar su entrega, así como la posibilidad de reservar productos que no se encuentren en el *stock* [2] físico de esta.

[1] **Amazon**: compañía estadounidense de comercio electrónico y servicios de computación en la nube
[2] **stock**: inventario, existencias de las que dispone la empresa o el comercio.

1.2. BuyYourself

BuyYourself es una *start-up* tecnológica que nació de la idea de agilizar los procesos que ocurren en las tiendas para poder mejorar la compra del usuario. Evitar o acortar colas, poder pedir productos desde la misma tienda si no tienen y poder gestionar los envíos a través de un dispositivo móvil o *tablet* son algunas de las funcionalidades en las que trabajamos.

La idea inicial era centrarse en el proceso de compra del usuario. Enfocarse al *BtoBtoC (Business to Business to Customer)*, es decir: crear una aplicación para los compradores mediante la cual estos pudieran entrar en la tienda, escanear los productos, añadirlos al carrito y pagar sin necesidad de pasar por caja.

Más adelante se decidió optar por un modelo más enfocado al *BtoB (Business to Business)*, es decir crear productos directamente para empresas de *retail*, en el cual se desarrollan una serie de módulos que permiten en su conjunto poder controlar los procesos referentes a compras, ventas, gestión de *stock* y envíos del comercio. Estos módulos incluyen una aplicación para compradores de nuestros clientes; un servidor que gestiona todos los datos y cálculos; una plataforma web para poder evaluar los datos de ventas y pedidos a la que llamamos *dashboard*; y finalmente la aplicación para vendedores, sujeto de este proyecto.

Actualmente nuestros potenciales clientes dentro del sector del *retail* son comercios o franquicias dedicadas a la venta de textil, ropa o complementos. No obstante es un modelo aplicable a otros tipos de *retailers*.

Nuestra centro de operaciones se sitúa en Barcelona Activa, donde tenemos la oficina. Desde esta trabajamos en nuestros ordenadores y resolvemos los problemas con comunicación constante. Las reuniones que tenemos para ir evaluando los progresos y el estado de los servicios, se desarrollan también en el entorno de Barcelona Activa.

2. Actores Implicados

2.1. Equipo BuyYourself

BuyYourself está compuesta por un equipo que cubre los ámbitos de programación tanto de *front-end* como de *back-end*, así como el diseño y la *UX*[3]. Dentro del campo de la programación, contamos con experiencia en *Android*, *iOS*, y Web.

2.2. El Inversor y los socios

BuyYourself cuenta con un inversor que dota a la empresa de recursos económicos para apoyar la inversión de los socios. Varios de los socios son también miembros del equipo. Los socios pusieron el capital inicial de la empresa, y en la ampliación inicial entró el inversor. El inversor, además, ayuda a la empresa con contactos.

2.3. Los clientes: empresas y comercios de retail

Principales clientes de nuestro modelo. Empresas que cuentan con tiendas físicas en varios puntos del territorio o varios países y que requieren de nuestros servicios para poder actualizar su negocio con nuevas tecnologías, aumentar beneficios y mejorar la experiencia de compra de sus consumidores. Principalmente empresas de tamaño medio ya que son las que más beneficios pueden sacar de nuestro modelo. Las grandes multinacionales tienen recursos como para crear sistemas muy específicos y a medida, y las pequeñas empresas con poas tiendas no sacarían suficiente rendimiento a la gestión entre ellas. Estos negocios han de disponer de un sistema *eCommerce* para gestionar su comercio online.

2.4. Director del proyecto y ponente

El director del proyecto es Luis Manuel Pérez Serna, miembro del equipo de *BuyYourself* y socio de la empresa. Centrado en el área de desarrollo *software* en *Android* y *iOS*, además de gestión del *back-end* y programación de servicios. Además se encarga de parte de la toma de decisiones del negocio.
La ponente del proyecto es María José Casany, del departamento de Eng.Serveis i Sistemes d'Informació de la Facultad de Informática de Barcelona.

2.5. Autor del proyecto

El autor del proyecto, Adrián Ruiz Cordón: estudiante de ingeniería informática desde el curso 2014-2015, especializándose en Ingeniería del Software y miembro de la plantilla de *BuyYourself* en calidad de estudiante en prácticas.

[3] **UX**: parte del diseño y desarrollo de *software* dedicada a la experiencia de usuario.

2.6. Usuarios: dependientes y trabajadores

Los principales usuarios de la aplicación son los empleados de las tiendas de nuestros clientes. Estos empleados gestionan las funcionalidades de la aplicación desde un dispositivo móvil *iPhone*.

3. Estado del Arte

Dentro del mercado de aplicaciones y servicios para empresas o comercios de venta al por menor, podemos encontrar bastante variedad de productos que cubren alguna de los aspectos que nosotros como empresa pretendemos mejorar y ofrecer a nuestros clientes.

3.1. Amazon Go

Amazon Go es un servicio de tiendas sin dependientes creado por *Amazon*. Mediante una aplicación instalada en el dispositivo *Android* o *iOS* del cliente, la persona puede entrar en una tienda, coger productos y salir sin pasar por caja. Se registran los productos que el cliente ha cogido de los estantes y una vez sale de la tienda; la aplicación cobra al usuario de la manera que haya registrado en esta. De momento solo se encuentra disponible en cuatro tiendas de Estados Unidos: tres en Seattle y una en Chicago y se centra en el sector de los alimentos. *Amazon* no supone un competidor actualmente dado que su modelo de negocio es propio para sus tiendas y productos, pero también está entrando en el sector del *retail*.

3.2. MishiPay

MishiPay[7] es una empresa que se dedica al *self-checkout* en Europa. Algunos de sus clientes destacados son *LeroyMerlin* y *MediaMarkt*, que ya funcionan con este sistema en algunas de sus tiendas físicas. *MishiPay* dispone de una aplicación móvil mediante la cual el usuario puede escanear los códigos de barras y añadir los productos a su cesta. Después, una vez ha decidido realizar la compra, se efectúa el pago a través de la aplicación y se desactiva el tag o identificador de seguridad del producto. Así el usuario puede salir de la tienda sin pasar por caja. Tiene un nicho de mercado más amplio en cuanto a *retailers*. No obstante, *MishiPay* sólo ofrece una herramienta a los clientes sin ningún tipo de personalización de cara a entrar en sintonía con la empresa.

3.3. NewStore

NewStore[8] es una empresa que opera en Europa y que ofrece a las empresas una manera de gestionar inventario y *omnicanalidad* a través de dispositivos como móviles o *tablets* con su app instalada. Es quizás la que más se asemeja a nosotros en cuanto a la parte del *Dashboard*, el servidor y la aplicación para comprador. Las principales funcionalidades que ofrecen el sistema de *NewStore* incluyen un *Dashboard* para controlar información de las ventas de la tienda y los pedidos, así como gestión de los pedidos online desde la tienda y las recogidas que se tienen que producir en físico. También acepta pagos en tarjeta desde su aplicación y desde cualquier punto del comercio. Diseña aplicaciones e infraestructuras específicas para cada cliente.

3.4. Mercaux

Mercaux[13] es una empresa que se dedica a dotar de plataformas digitales a los *retailers* para que sus dependientes puedan gestionar las ventas físicas online desde la tienda, así como proveer asistencia a los clientes. También permite la monitorización de datos y la integración de los servicios existentes en el comercio. Las ventas se pueden cerrar desde la aplicación de los vendedores a través de un proceso de pago con tarjeta.

3.5. Nextail

Nextail[14] es una empresa que se dedica a proveer un sistema de automatización y gestión de *stock* mediante inteligencia artificial y predicciones calculadas. De esta manera es capaz de mejorar el estado de los inventarios y producir soluciones para mejorar la distribución de productos entre los comercios del cliente.

3.6. Conclusiones

En conclusión, encontramos diferentes empresas que han visto la oportunidad de dedicarse a aplicar nuevas tecnologías en el sector del *retail*. La mayoría de ellas se centra exclusivamente en atacar un ámbito del problema como puede ser el *self-checkout* o la gestión de *stock*, proveyendo soluciones muy concretas para unos clientes muy específicos. Otro factor a tener en cuenta es el tipo de producto que ofrecen muchas de ellas. Ofrecen unas herramientas con poco margen de personalización y adaptabilidad a la infraestructura del cliente. Por otro lado, muchas de ellas no poseen propiedad intelectual de los productos que ofrecen, a diferencia de nosotros.

Nuestro sistema, en cambio ofrece una producto que integra soluciones tecnológicas actuales para resolver estos problemas, con una gran adaptabilidad a los clientes. Con ligeros cambios en nuestra estructura, podemos aprovechar nuestros servicios para diferentes *retailers* y dotarlos de unos servicios consistentes y fácilmente usables por los empleados de sus tiendas.

4. Formulación del problema

Según un estudio[4] realizado en 2015 por la consultora Coleman Parkes realizado para *Epson*, los españoles son los compradores que menos aguantan la espera en las tiendas. Un 36% de los clientes se va de la tienda sin comprar si ven una cola d. Señala además que la transformación del sector del *retail* con las nuevas tecnologías no solo conlleva importantes mejoras para los procesos internos, si no también será un aspecto crucial para evitar la pérdida de clientes. A nivel general el estudio refleja que los consumidores europeos consideran necesaria esta evolución del sector *retail* para mejorar la atención al cliente y ofrecer nuevas experiencias de compra. El estudio "*State of European Retailing*" concluye que a medida que la tecnología adquiere más relevancia en la vida de los consumidores, sus exigencias de cara a las experiencias de compra se centran cada vez más en un entorno adaptado a esta evolución.

Otro artículo a tener en cuenta de Epson fue realizado por el *Epson Blog Team*[5] para empezar a imaginar cómo sería la oficina o comercio en 2025. Constata que el principal factor que traerá una gran transformación en este sector será la personalización. El 72% de los encuestados que trabajan en el sector del *retail* señala que este sector podrá proporcionar mejores experiencias gracias a la tecnología y un 73% cree que los dispositivos móviles adoptan un papel cada vez más importante entre el cliente y la tienda. Este artículo señala también que la vertiente física del comercio debe evolucionar para poder seguir el ritmo al avance tecnológico. A la mitad de los encuestados les parece imposible que exista un futuro sin tiendas físicas. Los empleados europeos opinan que hasta 2025 un 56% de las decisiones de compra se tomarán en la propia tienda.

No obstante, el artículo de *Epson Blog Team* refleja también que un 63% de los encuestados cree que esta evolución tecnológica implica un reto en cuanto a costes, y el 40% opina que la falta de capacitación de los empleados para gestionar estas tecnologías puede tener un impacto negativo.

Es por eso que desde *BuyYourself* planteamos una solución fácil de gestionar por los empleados, que agilizará los procesos que suceden dentro de la propia tienda y que permitirá poder controlar todos los datos de los comercios de nuestro cliente de manera sencilla.

4.1. Objetivos principales

Nuestro principal objetivo es dotar a las empresas y comercios del sector del *retail* de herramientas tecnológicas para gestionar sus ventas y, sus envíos, evitar roturas de *stock* y generar correctamente las reservas; poder comparar y evaluar datos de beneficios, productos enviados, pedidos hechos; y además poder mejorar la experiencia de compra del cliente. Queremos poder facilitar la vida al comprador y a la vez la del consumidor.
Juntando las soluciones propuestas para estos problemas en un producto completo. Para ello planteamos una solución que consta de 4 servicios principales:

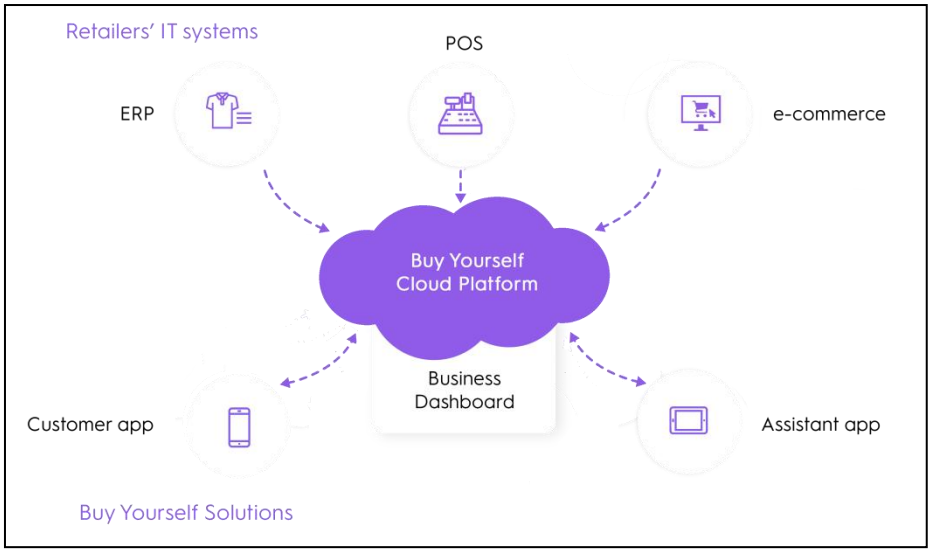


Figura 1: Esquema de los módulos BuyYourself

Aplicación de Comprador: Customer Experience App

La aplicación de comprador se centra en mejorar la experiencia de compra del cliente del comercio. Las principales funciones que ofrece son:

- *Express Checkout*: permite saltarse pasos al pagar usando información que el usuario ha introducido previamente o usado en compras anteriores.

- *Self-Checkout*: permite al usuario poder comprar los productos sin pasar por ninguna caja, eliminando así los tiempos de espera en la cola.

Aplicación de Vendedor: Assistant App Platform (AsApp)

Área en la que se centra el proyecto. Aplicación destinada a los dependientes de los comercios y empresas que contraten nuestros servicios. Esta aplicación permite poder gestionar productos, ventas y pedidos desde cualquier punto de la tienda.

Esta aplicación permite, a través del dispositivo del empleado poder acceder al catálogo y a información sobre los productos que hay en *stock*. Así mismo, puede consultar la disponibilidad de los productos en otras tiendas de la empresa y generar ventas físicas. Una vez realizada esta operación, se registrará el pago por parte del cliente en la misma aplicación desde cualquier punto de la tienda de manera segura y con diferentes canales habilitados: tarjeta o efectivo. Se enviará un ticket al cliente con la información de la compra, y así mismo se registrarán datos (siguiendo la LOPD Europea) para mejorar las estrategias de márquetin.

Además de esto, puede gestionar (aceptar o rechazar) pedidos *Click&Collect*, es decir: pedidos generados online destinados a recogerlos en una tienda física. El dependiente o dependienta tendrá acceso a la información de los productos solicitados para poder confirmar si están disponibles o no en la tienda para que el usuario los recoja. Por otro lado, se pueden generar pedidos online desde la misma tienda física en caso de que los productos no se encuentren disponibles en el stock propio. Se pueden generar reservas a otras tiendas de la franquicia que tengan stock. Esto crea una *omnicanalidad* perfecta para las franquicias, que pueden redirigir sus ventas online a tiendas físicas y ventas físicas al canal online.

Creando una aplicación sencilla de utilizar, completa y robusta conseguiremos que la curva de aprendizaje de los dependientes y las dependientas sea lo más ligera posible y pueda agilizar el buen funcionamiento interno de las tiendas; así como acercar aún más las nuevas tecnologías al sector del *retail*.

Business Dashboard

El *Dashboard* es una plataforma web mediante la cual el cliente puede ver reflejados los datos de sus comercios. Podrá ver el estado de las ventas, envíos y pedidos; así como el uso de las aplicaciones y el *stock* de sus tiendas. El *dashboard* permite también gestionar las tiendas remotamente incluso si tienen las aplicaciones funcionando.

Servidor Backend: Buy Yourself Cloud Platform

La plataforma de servidor que ofrecemos a los clientes permite conectar los servicios que ya utilizan las empresas (como por ejemplo el *eCommerce*) con el resto de servicios. De esta manera se podrán tener datos reales de los productos del catálogo, se podrán gestionar los datos para poder coordinar las diferentes tecnologías aplicadas en sus tiendas.

5. Alcance del proyecto y riesgos

La empresa cuenta ya con una versión para *Android* de la aplicación que se pretende desarrollar para *iPhone*. Esta aplicación contará con las funcionalidades básicas de la de *Android*. También cuenta con un *back-end* que es compartido con el resto de módulos del sistema, a través del que obtiene y envía la información pertinente.

Una de las funcionalidades principales que se desarrollarán serán la consulta del catálogo e información de los productos, donde se podrá ver el *stock* de los productos, su marca, su precio y las variantes disponibles. Otra funcionalidad con la que contará es la de generar reservas desde el terminal móvil, enviando un ticket al cliente con la información de esta reserva; y la capacidad de gestionar las reservas entrantes para preparar su entrega. Así mismo, también se podrán generar envíos online desde la tienda a través de la aplicación y pagarlos al momento; y también gestionar los pedidos entrantes. Además, la aplicación contendrá un sistema de autenticación de usuarios a través de mail y contraseña para cargar la información de cada tienda. Está planeado también que cuente con la capacidad de recibir notificaciones de pedidos recientes.

Una vez finalizado el desarrollo de la aplicación que contempla este proyecto y realizada la presentación final, se proseguirá el desarrollo de funcionalidades hasta que el producto cuente con las mismas que su versión *Android*. También se añadirán características como buscadores en las listas. La aplicación *iOS* será distribuida a los clientes que quieran operar en dispositivos *iPhone* y se les guiará para poder aplicar su uso en tiendas.

Esta aplicación y todos los módulos relativos que ofrecemos desde *BuyYourself* tendrán una vida útil larga siempre y cuando se vayan actualizando y adecuando a los requisitos del mercado. Al utilizar siempre las técnicas más novedosas y estar siempre a la última en cuanto a avances tecnológicos, hay pocas probabilidades de que el producto pierda repentinamente su valor.

Uno de los posibles riesgos que podría desembocar en el fin de la vida útil de nuestro modelos sería que el mercado sufriera un cambio drástico. Esto se podría dar si los comercios físicos dejaran de existir definitivamente. No obstante, al ser un cambio que se daría progresivamente, podríamos preparar un plan de actuación para incorporarnos al nuevo mercado o buscar alternativas.

También podría ocurrir que otra empresa con más recursos que nosotros desarrollara un modelo más completo y más eficiente con el que acaparara nuestro mercado y dejara obsoleta nuestra manera de actuar. Esto nos daría menos tiempo para desarrollar un plan de actuación o una alternativa, así que probablemente acabaría con la toma de una importante decisión: reorientar el rumbo de la empresa.

Por último, un riesgo potencial para este proyecto sería que todos nuestros clientes prefirieran la versión de *Android* para *tablets* que la versión *iOS* que se desarrolla en este.

6. Metodología

En *BuyYourself* apostamos por las metodologías ágiles. Dentro de las metodologías ágiles utilizamos *Scrum* [9], dado que los requisitos de nuestros proyectos van cambiando y necesitamos tener flexibilidad, a la vez que poder generar una buena productividad.

Tenemos *sprints* de 2 semanas que empiezan con un *Sprint Planning* donde decidimos qué historias de usuario vamos a tratar en durante el sprint.. Para llevar cuenta de esto utilizamos la plataforma *Jira*[10]. Cuando cerramos una tarea, antes de poder pasar al estado *Done* (finalizada), pasa por la revisión de uno o más compañeros para asegurarse que pasa los test necesarios, que cumple los requisitos de funcionalidad y que se adecua a lo esperado de su código (legible, reusable y eficiente). Una vez acabado el sprint, nos reunimos para hacer la sesión de *Close* del *Sprint*. En esta reunión discutimos que aspectos nos han costado más, revisamos las tareas que se han completado y analizamos donde nos hemos quedado con las historias de usuario que no se han cerrado, de cara a planear el siguiente *sprint*.

A mitad del *sprint*, tiene lugar una reunión que llamamos *Backlog Refinement* donde estudiamos como van las tareas y definimos las que podríamos incluir en el siguiente. Además, cada miembro del equipo escribe los requisitos de funcionalidad de sus tareas para revisarlos entre todos.

Además de las reuniones para planificación y revisión de los *sprints*, tenemos cada día una pequeña reunión (llamada *Daily*) donde comentamos que hizo cada uno el día anterior, qué avances se han logrado, si alguien está encallado en algún punto y a que va a dedicar su jornada cada miembro del equipo. Esta reunión se realiza observando el proceso de las tareas en *Jira* y actualizando las que sean necesarias.

Para la gestión del código, utilizamos repositorios de *BitBucket*[11]. Estos repositorios cuentan con varias ramas siguiendo la metodología *GitFlow Workflow*[12]: separamos entre una rama *master* (producción) donde se encuentra la versión actual del producto, ramas *hotfix* para poder solucionar pequeños *bugs*[4] en producción, ramas de *feature* para cada funcionalidad y una rama *develop* donde se suben las *features* una vez acabadas para hacer su revisión y corrección de pequeños errores.

Una vez un desarrollador acababa de implementar una funcionalidad, se creaba un *pull-request*, es decir, una solicitud para subir esa parte del código a la rama principal de desarrollo. Entonces otro miembro del equipo pasa una minuciosa revisión al código. Una vez corregidos los errores (si los hay), se acepta la *pull-request* y se integra la rama *feature* en la rama *develop*. Una vez integrada con éxito, se pasa a realizar pruebas funcionales para asegurar que todo va como tiene que ir.

[4] **bug**: error de software.

También se realizaron pequeñas reuniones al comienzo del desarrollo de cada una de la historias de usuario. Con esto se pretende que el estudiante y el tutor dentro de la empresa (el director del proyecto) tengan una mayor comunicación. Así se logra un desarrollo más guiado en función de los requerimientos de cada historia y una mejor mitigación de posibles desvíos. Además previene posibles bloqueos por parte del desarrollador.

Para plantear nuevas prácticas o mejoras y analizar nuestro proceder, hemos empezando a realizar retrospectivas y reuniones específicas sobre temas esenciales como los requisitos que debemos esperar de un código para que sea correcto o el proceder de la gestión de los repositorios.

Para los *mockups*[5] y el diseño de la interacción utilizamos la plataforma *Sketch* [15].

[5] **mock-up**: modelo a escala o tamaño real de un diseño o un dispositivo

7. Planificación Inicial

7.1. Calendario

En cuanto se puso en producción la aplicación para tablets *Android* (para uno de los clientes), el proyecto de la aplicación para *iOS* (sujeto de este trabajo final de grado) comenzó su desarrollo. Con el diseño ya realizado por el miembro del equipo que se encarga del UX, se comenzó el primer *sprint* del proyecto el 30 de julio de 2018. Los turnos de lectura de los trabajos finales de grado delante del tribunal para esta convocatoria son del 28 de enero al 1 de febrero. En base a que la memoria del proyecto debe estar una semana antes de la fecha del turno de lectura escogido para que el tribunal la pueda leer, se ha estimado que para el 21 de enero debe estar todo listo. Para que el 21 de enero esté todo el proyecto terminado, hará falta que la aplicación quede acabada con varias semanas de antelación. Se ha fijado la fecha para tener el producto acabado el 31 de diciembre. De esta manera, hay suficiente tiempo hasta la entrega de la memoria para hacer pruebas finales, redactar la memoria y revisar la documentación.

7.2. Sprints

El proceso de desarrollo se divide en *sprints*. Estos *sprints* duran dos semanas. Antes de empezar un sprint, más o menos hacia la mitad del sprint anterior, se hace una reunión de *backlog refinement*. En esta reunión, se mira de desgranar historias de usuario que se plantea tratar en el siguiente *sprint*. También se plantean los criterios de aceptación de estas.

Los *sprints* comienzan el lunes (con pequeñas excepciones debido a festivos o similares) con el *Sprint Planning*. En esta reunión de *Sprint Planning*, se establecerán las tareas que los miembros del equipo se comprometen a acabar antes del final. Se revisa la división de tareas y criterios de aceptación realizados en el *grooming* y se hace una estimación relativa de la complejidad de las historias de usuario.

Cuando finaliza el *sprint*, se realiza la reunión de *Sprint Close*. En esta reunión tienen lugar la validación final de las tareas acabadas, una pequeña demostración de lo que ha hecho cada uno y se hace también un análisis sobre la velocidad (en base a las historias cerradas y sin cerrar). Se hace una reunión interna en el que uno de los miembros del equipo toma el rol de *Product Owner* [6]para valorar el estado del producto.

7.3. Tareas

Se pueden distinguir diferentes tipos de tareas relativas a cada parte del proceso de realización del proyecto en función de los aspectos que tratan. Hemos planteado agruparlas para llevar un mejor control.

[6] **Product Owner**: Rol que representa al cliente que contrata al equipo para desarrollar un producto

7.3.1. Planificación y marco de trabajo

La parte de planificación engloba tanto la parte de incepción del producto y preparación del entorno, como todo lo que atañe a planificar los diferentes *sprints*. Una vez el diseño estuvo hecho por parte del encargado de *UX*, tuvimos una reunión correspondiente a la Incepción del Producto. Durante esta reunión se discutieron las tareas que se iban a llevar a cabo. Se decidieron las historias de usuario que se iban a tratar y el alcance del producto. Se fijó en qué punto estaba el *MVP* (*Mínimum Value Product*), el punto en el que el producto cumple las expectativas y las funcionalidades deseadas; y las posibles mejoras así como funcionalidades extra de cara a las futuras versiones.

Dentro de las tareas de planificación entrarían también la definición y preparación del entorno de desarrollo en el que se va a trabajar. Estas tareas se cerraron durante los tres primeros sprints del proyecto, en los cuales se fue preparando la estructura que iba a seguir la aplicación y se fueron configurando las herramientas de trabajo; a la vez que se estudió el lenguaje de programación y las herramientas disponibles. Esta tarea fue evolucionando conforme aumentaban las necesidades de desarrollo hasta tener un marco de trabajo completo.

7.3.2. Desarrollo

Para tal de ordenar mejor las áreas de la aplicación en las que se va a trabajar y así poder establecer una planificación mejor, se han separado las tareas de desarrollo en grupos de historias de usuario. Así podemos establecer una relación de precedencia entre las tareas. Está previsto que el desarrollo finalice para fin de año (Sprint 11) para poder hacer pruebas exhaustivas sobre la aplicación y revisar *bugs*.

LogIn

El grupo de historias de usuario de *login* incluye el desarrollo de la pantalla y la adaptación de los servicios necesarios a la plataforma *iOS*. Fue lo primero en desarrollarse, dado que es una funcionalidad simple y puede utilizarse para estimar el coste temporal del resto. Estas tareas se cerraron durante el primer *sprint*.

Catálogo y productos

Este grupo de historias de usuario incluye las funcionalidades referentes al catálogo de productos de la aplicación. Todas las que tengan que ver con maquetar las pantallas y poder adaptar el servicio que recupera el catálogo, así como la información de los productos. Sin estas tareas no podemos empezar a programar el resto de la aplicación dado que es la parte que gestiona todo lo referente a la información de los productos que tratamos. Aquí tenemos navegar por las categorías del catálogo y poder ver los productos de la categoría. También incluye la parte de mostrar la información del producto seleccionado en el catálogo. Ha sido la siguiente en desarrollarse. Las funcionalidades que aquí se trataron se han cerrado entre el sprint 2 y el sprint 5.

Reservas

Este grupo de historias de usuario hace referencia a los procesos de reserva de un producto y la confirmación o rechazo de esta en función de la disponibilidad en *stock* de la tienda destino. Incluye servicios y maquetación de pantalla. Se ha comenzado su desarrollo en el *sprint* número 5 y se prevé acabarlo para el *sprint* 6.

Cesta online

Este grupo de historias de usuario incluye todo lo referente a realizar ventas *online* desde la tienda: añadir productos a la cesta *online*, confirmar, realizar el pago y enviar el ticket correspondiente al cliente. Implica también la selección de métodos de envío y la introducción de los datos de envío del cliente Aquí se realizará maquetación de las pantallas y los servicios que hagan falta. Estas tareas se llevarán a cabo entre el *sprint* 6 y el *sprint* 9, ya que es de las que implica más funcionalidades y es uno de los puntos fuertes de la aplicación.

Click&Collect (Recogida en tienda)

Este grupo de historias de usuarios se centra en poder aceptar o rechazar pedidos que se hayan hecho para recoger en tienda según se tengan los productos o no, e informar al consumidor. Incluye servicios y maquetación de pantalla. Estas tareas comenzarán en el *sprint* 9 y se planea acabarlas en el *sprint* 10.

Notificaciones y ajustes

Estas tareas incluyen las historias de usuario que tratan las notificaciones al recibir un pedido o reserva, así como la pantalla de ajustes. De momento, en este proyecto solo entrará como ajustes el cierre de sesión. Estas tareas serán las últimas en desarrollarse dado que no son muy complejas y son añadidos a las anteriores. Por ello se les destinará el *sprint* 11.

7.3.3. Documentación

Las tareas de documentación incluyen principalmente las entregas de GEP y la documentación final; además de la documentación pertinente extraída de las reuniones que vaya manteniendo el equipo. La documentación final irá evolucionando conforme evolucionen las entregas de GEP, mejorando con el *feedback* recibido. Una vez acabada la aplicación y revisada, se pasará a revisar y acabar la memoria durante los *sprints* 12 y 13.

7.3.4. Pruebas y revisión

A parte de las pruebas que va realizando el estudiante a la par que desarrolla nuevas funcionalidades, se realizan pruebas de validación con unos criterios previamente establecidos. Una vez se acaba una tarea, se solicita a un miembro del equipo (en la mayoría de casos el director del proyecto) que revise el código y la funcionalidad. A esto se le suelen dedicar unas 3 horas por *sprint*. Además, al final del *sprint* se hace una demo de lo que ha trabajado cada uno y se revisa. Durante el *sprint* 12 se realizarán pruebas más exhaustivas de cara a la versión final.

7.4. Estimación de tareas y dependencias

A continuación se estimarán las horas empleadas o que se emplearán en el proyecto. Para las tareas de desarrollo, planificación, marco de trabajo y pruebas (que suceden dentro del horario laboral) se contabiliza 4h por día (jornada laboral siguiendo el convenio de la FIB [16]).

Tarea	h. Estimadas	Dependencias
1) Planificación	12	
2) Marco de trabajo	28	
3) Desarrollo: LogIn	20	
4) Desarrollo: Catálogo+Info productos	96	3)
5) Desarrollo: Reservas	40	4)
6) Desarrollo: Cesta online	120	4)
7) Desarrollo: Click & Collect	60	4)
8) Desarrollo: Notificaciones y ajustes	40	3)
9) Pruebas	40	3), 4), 5), 6), 7), 8)
10) Documentación GEP	78	1)
11) Documentación final	20	10)
12) Revisión de documentación final	20	11)
13) Preparación de la presentación	20	10)
14) Revisión por parte de miembro del equipo	36	
15) Planificación de sprints	72	
Total	**702**	

Tabla 1: Estimación de horas y dependencias**.**

7.5. Recursos

Para el desarrollo de la aplicación y el proyecto se están utilizando recursos de varios tipos: personales, materiales y *software*.

Personales

En los recursos personales entran las personas implicadas en el desarrollo del proyecto. Aquí tenemos al estudiante y al equipo de *BuyYourself*, encargado de darle soporte y supervisar las tareas que se van realizando, así como establecer criterios de aceptación para estas. En cuanto al proceso de la redacción de la memoria, también entran la profesora ponente y el tutor de GEP. Estos dos se encargan de atender las dudas que puedan surgir sobre la memoria.

Materiales

En los recursos materiales entraría la oficina de *BuyYourself*, que se encuentra en Barcelona Activa (Carrer Llacuna, Barcelona). A parte de esto, intervienen elementos *hardware*. Estos elementos *hardware* son:

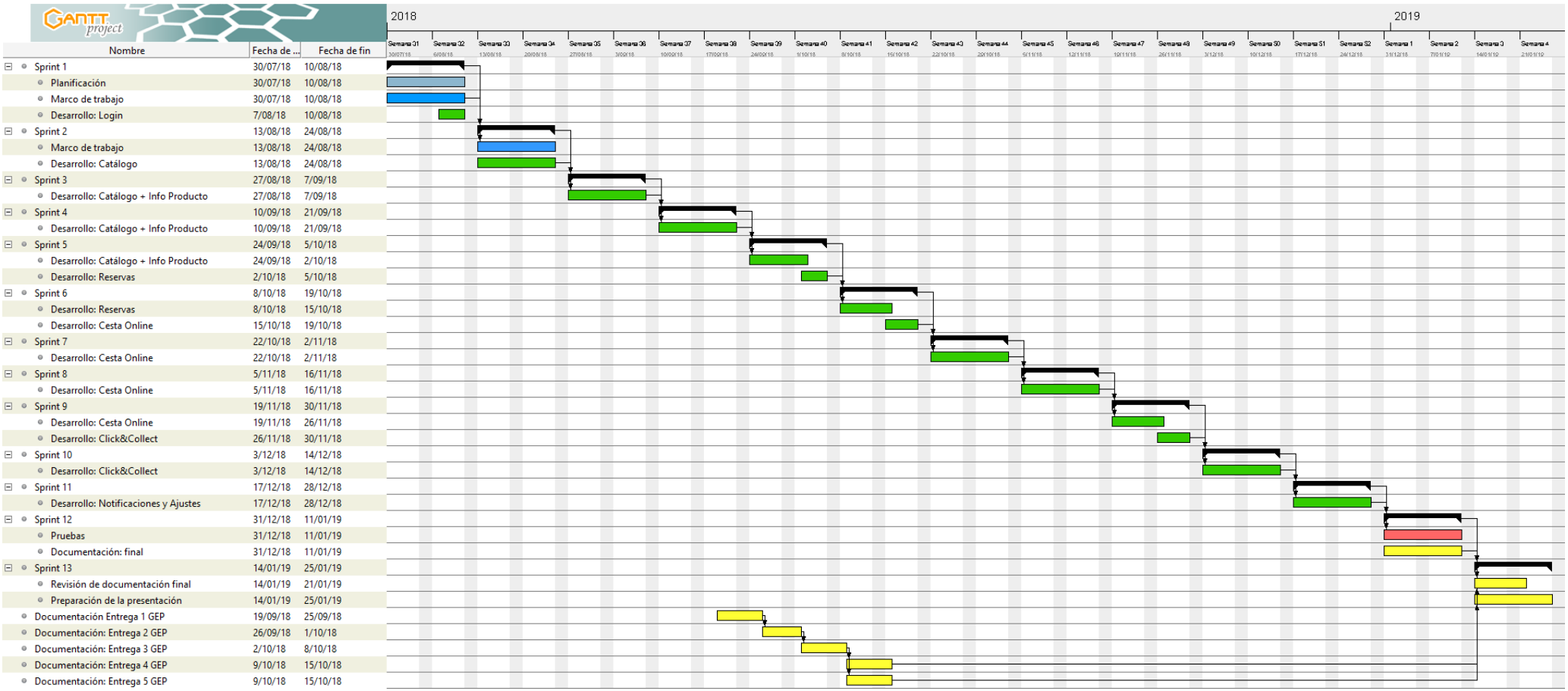
- **Ordenador de torre Mac**: desde donde se desarrolla la aplicación, con el entorno de trabajo adecuado instalado y preparado. Cedido por uno de los socios de la empresa.
- **Ordenador de torre auxiliar:** ordenador con un entorno MAC instalado a través de una máquina virtual para seguir el desarrollo de la aplicación una vez el socio que prestó el ordenador MAC quiera recuperarlo.
- **Dispositivo tablet Android**: dispositivo con la versión *Android* instalada para comprobar la interoperabilidad con la aplicación *iOS*.
- **Dispositivo móvil iPhone** (superior a *iPhone* 7): dispositivo desde el cual se efectuarán las pruebas de interacción y las demostraciones pertinentes para certificar el buen funcionamiento de la aplicación.
- **Servidor *cloud***: servidor que controla los datos y gestiona los procesos de *back-end*. Con él se comunicarán los servicios de la aplicación para recibir y transmitir la información necesaria.

Software

Para el desarrollo del proyecto y la memoria se están utilizando las siguientes herramientas *Software*:

- **XCode**: entorno de desarrollo de la aplicación en *iOS*, preparado para programar en *Swift*.
- **Jira**: *software* de seguimiento de proyectos e incidencias a través del cual se planifican los *sprints* y se gestionan las historias de usuario.
- **Bitbucket**: servicio de alojamiento de repositorios donde se suben las versiones de la aplicación y el código.
- **SmartGit**: aplicación de gestión de repositorios que conecta con *Bitbucket* para gestionar las ramas siguiendo el esquema *GitFlow* (mencionado en la anterior entrega).
- **Sketch**: plataforma destinada al diseño de pantallas e interacción de aplicaciones.
- **Abstract**: plataforma para controlar las versiones del diseño de la aplicación, funcionamiento similar a repositorios *git*.
- **Robomongo**: herramienta de gestión y consulta para bases de datos en *mongoDB*.
- **Slack**: herramienta de comunicación a través de la cual se comunica el equipo de *BuyYourself*.
- **Google Drive y Google Docs**: sistema de almacenamiento en la nube donde se guarda documentación relativa al proyecto. Con *Google Docs* se editan documentos y tablas para almacenar información sobre el proyecto.
- **Microsoft Word**: programa de edición de documentos para redactar las versiones finales de los entregables y la memoria.
- **Virtual Box**: *software* de virtualización utilizado para poder utilizar un entorno MAC en el ordenador de oficina.

7.6. Diagrama de Gantt



Nombre	Fecha de ...	Fecha de fin
Sprint 1	30/07/18	10/08/18
Planificación	30/07/18	10/08/18
Marco de trabajo	30/07/18	10/08/18
Desarrollo: Login	7/08/18	10/08/18
Sprint 2	13/08/18	24/08/18
Marco de trabajo	13/08/18	24/08/18
Desarrollo: Catálogo	13/08/18	24/08/18
Sprint 3	27/08/18	7/09/18
Desarrollo: Catálogo + Info Producto	27/08/18	7/09/18
Sprint 4	10/09/18	21/09/18
Desarrollo: Catálogo + Info Producto	10/09/18	21/09/18
Sprint 5	24/09/18	5/10/18
Desarrollo: Catálogo + Info Producto	24/09/18	2/10/18
Desarrollo: Reservas	2/10/18	5/10/18
Sprint 6	8/10/18	19/10/18
Desarrollo: Reservas	8/10/18	15/10/18
Desarrollo: Cesta Online	15/10/18	19/10/18
Sprint 7	22/10/18	2/11/18
Desarrollo: Cesta Online	22/10/18	2/11/18
Sprint 8	5/11/18	16/11/18
Desarrollo: Cesta Online	5/11/18	16/11/18
Sprint 9	19/11/18	30/11/18
Desarrollo: Cesta Online	19/11/18	26/11/18
Desarrollo: Click&Collect	26/11/18	30/11/18
Sprint 10	3/12/18	14/12/18
Desarrollo: Click&Collect	3/12/18	14/12/18
Sprint 11	17/12/18	28/12/18
Desarrollo: Notificaciones y Ajustes	17/12/18	28/12/18
Sprint 12	31/12/18	11/01/19
Pruebas	31/12/18	11/01/19
Documentación: final	31/12/18	11/01/19
Sprint 13	14/01/19	25/01/19
Revisión de documentación final	14/01/19	21/01/19
Preparación de la presentación	14/01/19	25/01/19
Documentación Entrega 1 GEP	19/09/18	25/09/18
Documentación: Entrega 2 GEP	26/09/18	1/10/18
Documentación: Entrega 3 GEP	2/10/18	8/10/18
Documentación: Entrega 4 GEP	9/10/18	15/10/18
Documentación: Entrega 5 GEP	9/10/18	15/10/18

7.7. Posibles desviaciones

Por lo general no hemos detectado grandes posibles desviaciones. Los impedimentos que nos pueden traer más problemas son tecnológicos. La tecnología avanza y cambia, lo cual genera una cierta incertidumbre. Pues si alguna de las tecnologías que utilizamos fuera sustituida o actualizada, deberíamos dedicar tiempo a adaptar el proyecto a esta. También podríamos tener pequeñas desviaciones temporales debidas a tareas mal estimadas, pero esto es fácilmente mitigable con las reuniones de los *sprints* y soporte del equipo.

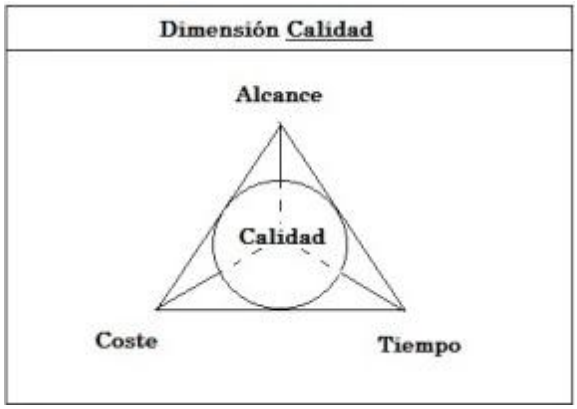


Figura 2: Triángulo de Hierro

Para mitigar grandes posibles desviaciones, hemos planteado un plan de actuación en base al Triángulo de Hierro de la metodología *Agile*. Hay tres principales factores para definir la calidad de un proyecto: alcance, tiempo y coste. Para maximizar la calidad, se suelen fijar dos de estos para el desarrollo, de tal manera que queda un tercer factor que puede variar en función de las desviaciones.

En este caso nuestros factores fijos son el tiempo (limitado a la entrega del proyecto con los las fechas límite comentadas en el primer apartado) y el coste (los recursos). Por lo tanto si las desviaciones supusieran un gran riesgo para la calidad del proyecto, se reduciría el alcance de este en similar medida.

8. Gestión económica y sostenibilidad

Para poner en contexto el trabajo de análisis de sostenibilidad, partimos del resultado de haber realizado la encuesta[7]. No es la primera vez que hago un análisis del impacto de un proyecto y su sostenibilidad. En asignaturas pasadas como Viabilidad de Proyectos Empresariales o Proyecto de Ingeniería del Software tuvimos que analizar costes y huella ecológica. No obstante, no conocía todas las herramientas y conceptos que se presentan en la documentación de GEP.

Por un lado, al estar bastante al tanto de actualidad en cuanto al impacto social de las nuevas tecnologías siendo contemporáneo a ellas, no me es tan complicado hacer un análisis y poder razonar sobre la accesibilidad de estas a la sociedad. Por otro lado, el tema de la dimensión de los costes, sí que me es menos familiar, dado que a pesar de haberlo trabajado en otras asignaturas, no es algo que utilice día a día.

En cuanto a habilidades personales, me considero una persona con bastante facilidad para proponer ideas y soluciones. Me cuesta poco mirar el problema desde varios puntos de vista para poder identificar los factores. Por ello, puedo proponer soluciones que cumplan con su cometido y que ayuden a maximizar un impacto positivo. En el caso que nos atañe, el sector *TIC*, me veo bastante capaz de dar soluciones que impacten positivamente en el avance de la sociedad minimizando los impactos negativos en esta.

8.1. Estudio de la dimensión económica

8.1.1. Costes directos

Dentro de los costes directos que tenemos en cuenta a la hora de calcular la estimación del presupuesto para la dimensión económica, podemos separar principalmente entre Recursos Humanos y Recursos Materiales.

Recursos Humanos

En el apartado de recursos humanos, contamos y estimamos el coste de las horas que dedicará cada miembro del equipo implicado en el proyecto. Por un lado tendremos la faena propia del estudiante, la cual se paga a 8€ la hora. La jornada de trabajo al ser estudiante en prácticas es de 4h diarias, siguiendo el convenio de estudiantes de Grado en prácticas de empresa de la Facultad de Informática de Barcelona[16].

[7] goo.gl/kWLMLE

Para el cálculo del coste, hemos de tener en cuenta también que la parte de documentación del proyecto, ya sea redacción de las entregas o de la memoria final; se hace a parte de la jornada laboral y que, por tanto, no influye en el coste. Teniendo en cuenta estas consideraciones y las tareas descritas en la planificación temporal, la estimación del coste personal quedaría de la siguiente manera:

Tarea	Horas estimadas	Coste estimado
Planificación y marco de trabajo	40	320 €
Desarrollo	376	3.008 €
Pruebas	40	320 €
Total	**456**	**3.648 €**

Tabla 2: Coste personal del estudiante

A parte del coste del trabajo del estudiante, hemos también de considerar el coste del resto de miembros del equipo que intervengan en el proyecta. Para ello calculamos el coste de las horas invertidas por el miembro del equipo encargado de hacer la revisión del desarrollo en cada *sprint* (lo cual suma aproximadamente 3h por s*print*) y las horas invertidas en planificación de las tareas del proyecto en los diferentes *sprints* por parte de todo el equipo (lo cual viene a ser aproximadamente 2h por Sprint). El sueldo aproximado de los miembros del equipo (al ser autónomos y socios, varía y no es fijo) se puede aproximar a unos 20€ la hora. Contamos 3 miembros del equipo que asisten a las reuniones de planificación de los *sprints*, así que sumaremos sus horas para estimar el coste.

Tarea	Horas estimadas	Coste estimado
Revisión	36	720 €
Planificación de *sprints*	72	1.440 €
Total	**108**	**2.160 €**

Tabla 3: Coste personal del resto del equipo

El resto de horas reflejadas en la planificación inicial y no contabilizadas en estos cálculos han sido horas que el estudiante ha dedicado fuera de horas de trabajo en su casa, por lo que no tienen coste económico. Aquí se incluirían las horas dedicadas a la documentación de GEP, la documentación final, la revisión de esta y la preparación de la presentación final.

Recursos Materiales

En cuanto a recursos materiales, hemos de empezar contando con el ordenador de sobremesa donde se ha comenzado el desarrollo y sus periféricos. El ordenador es un *Mac OS High Sierra* que tiene ya 8 años, cedido por uno de los socios de la empresa. El equipo está ya totalmente amortizado. Por su lado, contamos con un ratón, un teclado y unos cascos de baja gama. También disponemos de un monitor para el ordenador de sobremesa. Los mismos periféricos se han utilizado para el ordenador donde se ha proseguido el desarrollo una vez el socio que prestó el MAC requirió de él. Es bastante nuevo (se compró en junio) y se prevé que sea amortizado dado que es parte del entorno de trabajo que ocupa el estudiante. Se tuvo que adquirir RAM adicional para poder lanzar el entorno MAC en este ordenador (incluida en el subtotal de la tabla correspondiente).

Por su lado, el teléfono *iPhone 7* o superior que se utilizará para las pruebas, será adquirido en el mercado de segunda mano o cedido por algún familiar o conocido del estudiante, así que de momento no se puede evaluar su coste. La *tablet* que se usará para las pruebas de interoperabilidad será una de las disponibles dentro de la empresa para hacer las pruebas de la aplicación *Android*. Tenemos diferentes modelos de *tablets*, así que su coste dependerá de cuál esté disponible en el momento que de ella se requiera para hacer las pruebas.

Recurso	**Precio**
Ratón	10€
Teclado	10€
Cascos	10€
Monitor	110€
Ordenador de sobremesa	450€
iPhone	350€
Tablet Android	200€
Total	**1.140€**

Tabla 4: Costes materiales

8.2. Costes indirectos

De los costes indirectos del mes tenemos que contar que su uso no es exclusivo del desarrollo del proyecto. Por eso se ha hecho un cálculo aproximado de cuál es su coste mensual para la empresa. Dentro de estos costes se incluirían:

Recurso	Coste mensual
Electricidad	65,50€
Internet	60,50€
Alquiler oficina (Barcelona Activa)	329€
Servidor cloud	83€
Software Jira	10€
Software BitBucket	10€
Total	**558€**

Tabla 5: Costes indirectos

Otro coste indirecto a tener en cuenta es la tarjeta *T-jove*[8] de 2 zonas que emplea el estudiante para acudir a la oficina cada día. Se trata de una tarjeta trimestral con viajes ilimitados que cuesta **142€**. Este coste lo asume el estudiante cada 3 meses, dado que es además una tarjeta que utiliza en su día a día para otros desplazamientos.

Para calcular el subtotal correspondiente a este proyecto, hemos de contar que estos recursos indirectos son usados por los cuatro miembros actuales del equipo, así que su gasto queda más repartido. Contabilizaremos la parte correspondiente al estudiante (1 de 4 miembros). Teniendo en cuenta que la duración del proyecto es de 6 meses (de Agosto a finales de Enero), esto no da la cifra de **837€.**

8.1.3. Imprevistos y contingencia

Los imprevistos que se pueden dar vienen directamente relacionados con las desviaciones comentadas en la planificación temporal. Se puede dar el caso de que una nueva tecnología entre en el mercado y tengamos que readaptar el producto. Esto supondría primero un gasto en horas respecto a adaptarse al nuevo marco de trabajo y posteriormente un gasto en horas de refactorización o desarrollo. Para tal de cubrir esto, vamos a destinar un importe similar a lo que costaría a otro miembro del equipo realizar un 10% de las horas destinadas al proyecto. De esta manera, podríamos cubrir esta necesidad con ayuda de otro desarrollador. Serían unas 70 horas aproximadamente, y dado que su coste es de 20€/h al ser un miembro de la plantilla, asciende a **1.400 €**.

[8] **T-Jove**: Tarjeta trimestral de viajes ilimitados para jóvenes de entre 15 y 25 años. Más información en https://www.ambmobilitat.cat/Billetes/TipoBillete.aspx?tipo=7

Por otro lado hay imprevistos que sí que podemos analizar y cuantificar para poder destinar una partida de dinero que están relacionados con el *hardware*. De hecho uno de ellos ya ha sido analizado y controlado. El socio que prestó el MAC donde se estaba desarrollando el proyecto lo ha puesto a la venta, así que hemos tenido que destinar una partida de dinero de para poder añadir más RAM al ordenador de sobremesa que utilizaba antes el estudiante para poder arrancara un entorno MAC virtualizado. Dado que es un gasto que ya se ha asumido y controlado, este se incluye en el subtotal de los recursos materiales. Pero se dedicarán unos **400 €** (precio aproximado del resto de componentes del ordenador sin contar la nueva RAM) por si se requiriera actualizar alguno de ellos.

Finalmente, podríamos entrar a valorar imprevistos relacionados con cortes de luz, incendios o inundaciones que pudieran afectar al mobiliario y *hardware* de la oficina. No obstante, si esto sucede, el seguro que debe tener la empresa cubriría estos gastos.

8.1.4. Presupuesto total

Recurso	Coste
Recursos Humanos Estudiante	3.648 €
Recursos Humanos Equipo	2.160 €
Recursos Materiales	1.140 €
Costes Indirectos	837 €
Presupuesto de contingencia	1.800 €
Total	**9.585 €**

Tabla 6: Presupuesto total

8.1.5. Control de gestión

Para poder establecer un procedimiento de control de gestión hemos de poder valorar en qué puntos podemos descubrir los posibles fallos de control en cuanto a presupuesto. Uno de los primeros factores podría ser la diferencia de horas de trabajo reales respecto a las estimadas. En la estimación de horas no se han tenido en cuenta días festivos o eventualidades como asistencia a conferencias o ferias de tecnología. Para poder controlar esto, desde el inicio de las prácticas, el estudiante lleva un seguimiento de las horas reales que trabaja cada semana y las tareas que realiza. Así mismo, por parte de la empresa también se lleva un control de las horas que realizamos tanto yo como el otro estudiante en prácticas. Podríamos utilizar el indicador de Desviaciones en la realización de las tareas (en horas).

En cuanto al seguimiento y la gestión que se lleva desde la empresa, nos guiamos por la **velocidad de los *sprints***. En cada *sprint* se trata un cierto número de historias de usuario. Estas historias de usuario tienen un número asignado de puntos de historia, en base a lo que el equipo ha estimado que costará llevarlas a cabo. Cuando se cierra un *sprint*, se calculan los puntos de historia de usuario planeados y los que finalmente se han cerrado. Conforme van avanzando los *sprints* y se tienen datos sobre la velocidad, se puede ir planteando si se deben realizar cambios en el ritmo para adecuarlo al desarrollo o si se debe modificar la planificación. Este dato se suele utilizar en la mayoría de proyectos de metodología ágil para poder analizar el proceso de desarrollo.

Además de este dato, *Jira*nos aporta estadísticas y datos sobre el total de historias resueltas y las que nos quedan por resolver, sus puntos de historia y el tiempo en que se han resuelto. Nos da perspectiva sobre donde estamos, a donde queremos llegar y si vamos por buen camino.

8.1.6. Reflexión sobre la dimensión económica

Al ser un proyecto enfocado sobre todo al desarrollo *software*, el coste en cuanto a materiales es bastante reducido, con lo que pueden darse pocos casos en los que un mal control de estos costes materiales influya negativamente en el coste final del proyecto. Por otro lado, los costes humanos del proyecto sí que suponen un factor a tener en cuenta. Se podría realizar este proyecto en menos tiempo si otros miembros del equipo se implicaran más directamente en su desarrollo, pero aquí intervendría también el coste que esto supondría. Un miembro del equipo con contrato de trabajador (que no sea de prácticas) cobra sustancialmente más que el estudiante. Se podría estimar la diferencia respecto horas y precio que supondría añadir otro trabajador, pero esto supondría dejar de lado otros proyectos de la empresa y podría suponer pérdidas de capital. Además, la aplicación que se desarrolla con este proyecto pretende ser la base a través de la cual comercializar una parte de nuestro modelo, así que no se pueden descuidar las otras partes o módulos en los cuales trabaja el resto del equipo.

En cuanto al impacto económico que puede suponer la implantación de nuestro sistema en los comercios, hemos de tener en cuenta lo comentado en el estado del arte. Actualmente hay muchas empresas que no cuentan con una adaptación tecnológica de sus infraestructuras; así que las posibilidades de mejora son muy elevadas. Esto se reflejaría en sus beneficios, como ya se están reflejando con la empresa Misako, empresa que ya cuenta con parte de nuestro sistema en sus tiendas. No se pueden compartir los datos exactos por motivos de contrato, pero los envíos desde tienda han incrementado sustancialmente sus ventas y beneficios.

8.2. Estudio de la dimensión ambiental

Para analizar el impacto ambiental de este proyecto, debemos tener en cuenta los factores que pueden afectar al medio ambiente. Por un lado tenemos los recursos propios que utilizan la empresa y el estudiante, como la repercusión en uso de recursos de los negocios de nuestros clientes.

Impacto de la empresa

En cuanto a los recursos de la empresa, tenemos que tener en cuenta sobre todo el gasto energético. Nuestra oficina de Barcelona activa está dotada de iluminación y calefacción, al igual que el resto de las oficinas. No obstante, el consumo de estos recursos no variará en función del proyecto, ya que en la oficina se llevan a cabo las diferentes tareas de la empresa.

Respecto al consumo de los ordenadores de la empresa, podríamos razonar de la misma manera en cuanto a si influye o no el desarrollo del proyecto. No obstante, hay una pequeña diferencia: el ordenador *MAC* desde el que se desarrolla el proyecto ha sido dejado al estudiante por uno de los socios de la empresa. Este ordenador ha sustituido al ordenador con el que empecé las prácticas, ergo el consumo no se suma al del anterior ordenador. El número de terminales sigue siendo el mismo.

Por otro lado, el ordenador personal desde donde se redacta casi toda la documentación del proyecto es mi ordenador portátil personal, adquirido hace 4 años y completamente amortizado durante los cursos de la carrera.

Otro tipo de recursos que podrían incluirse como material de oficina podrían ser las hojas empleadas en tomar anotaciones sobre el proyecto (cosas a revisar, dudas que comentar, etc). No obstante, se hace un gran aprovechamiento del papel, y al ser un proyecto tecnológico, casi toda la información está en formato digital.

En cuanto al dispositivo *tablet Android* destinado las pruebas de interoperabilidad, será utilizado uno de los que se usa para el desarrollo y mantenimiento de la aplicación *Android*, propiedad de la empresa y totalmente amortizado durante el periodo de desarrollo anterior a comenzar este proyecto. Por su lado, el dispositivo *iPhone* desde el que se realizarán las pruebas de la aplicación, será un dispositivo que o bien se pedirá prestado o se adquirirá en el mercado de segunda mano. Por lo tanto, esto afecta de manera positiva al impacto ambiental, dado que se reutilizará un dispositivo alargando su vida útil y sin necesidad de adquirir otro nuevo, con el gasto de recursos que implica.

Además de lo mencionado anteriormente, tenemos un elemento muy importante: la estructura de los proyectos que desarrolla la empresa. Con la estructura que utilizamos, podemos reutilizar muchos recursos (como el servidor o aplicaciones que ya se hayan desarrollado), lo cual reduce el impacto que podría suponer duplicar estos recursos; ya sea en cuanto a consumo energético o en infraestructura.

Impacto de los clientes

Los clientes necesitarán dispositivos *iPhone* para que los empleados y las empleadas de sus tiendas puedan utilizar la aplicación. Esto puede suponer un impacto medioambiental mayor, dado que si es una gran empresa con muchas tiendas, necesitará una gran cantidad de dispositivos. Estos dispositivos, si posteriormente y una vez hayan agotado su vida útil no son reciclados o desechados de la manera correcta, pueden suponer un gran problema para el medio ambiente y la huella ecológica.

Por otro lado, un buen punto positivo de este proyecto es que podrá aumentar la eficiencia de los transportes de pedidos. Al tener un sistema de gestión de *stock* unificado, podemos redirigir la preparación de los envíos a la tienda que tenga los productos necesarios. De esta manera, evitamos que se tenga que hacer transporte desde varias tiendas para preparar un único pedido. Por lo tanto, reducimos el CO2 producido por este proceso, ya que cuantos menos viajes se tengan que hacer, menos emisiones tendremos. Aún así, es un factor que se puede mejorar más, dado que es relativamente leve si tenemos en cuenta la cantidad de envíos que se generan.

Otro factor a tener en cuenta es el de los tickets. Al permitir enviar tickets digitales a los compradores de los comercios de nuestros clientes, reducimos de manera drástica el consumo de papel de estos.

8.3. Estudio de la dimensión social

Para analizar el impacto social de la aplicación, tenemos que partir de la situación comentada en el apartado donde se realizó el análisis del estado del arte y la contextualización. Pese al receso económico de los últimos años, siguen proliferando negocios de venta al por menor, ya sean pequeños comercios o grandes cadenas. Actualmente muchos consumidores buscan pasar menos tiempo en las tiendas comprando, dado que en la sociedad actual vivimos muy deprisa. Queremos tenerlo todo rápidamente y sin muchas complicaciones, para poder disponer de tiempo para hacer muchas cosas. En este ámbito, el proyecto aporta una solución que ayudará a tener mejor control de lo que pasa en la tienda, y así minimizar problemas.

Se puede tender a pensar que con este modelo de aplicación, los puestos de trabajo de los dependientes de las tiendas están en peligro. Pero no es así. Los trabajos dentro de las tiendas evolucionarán con nuestra aplicación. No tendremos a los dependientes y dependientas de siempre que están en la caja todo el rato, si no que ahora podrán ejercer la función de cobrar al usuario desde varios puntos de la tienda.

Otro factor a tener en cuenta es la compra *online*. El comercio *online* aumenta cada vez más, pues ahorra tiempo de ir a la tienda física a mirar los productos y comprarlos. No obstante, mucha gente sigue prefiriendo comprar en físico o al menos acercarse a la tienda a ver los productos. Además, en las tiendas físicas también tenemos asesoramiento y atención por parte de los y las dependientes. Por ello, la capacidad de gestionar los pedidos online desde la tienda a través de la aplicación, tendrá un impacto positivo en la experiencia de compra del usuario y su visión de los comercios de *retail*.

En cuanto a la posibilidad de pedir online para recoger en la tienda, también soluciona otro problema. Muchos compradores no están siempre en casa para poder recoger el envío. Trabajan, hacen recados, etc. Por lo tanto, al facilitar la recogida en tienda, el cliente puede gestionarse mejor y organizarse para poder recoger los productos cuando estén y en el comercio que mejor le vaya. Además, con el control de *stock* que se realiza gracias a la aplicación y nuestros algoritmos, podemos asegurar la mejor manera de recoger todos los productos que ha pedido en el mismo sitio.

En cuanto al factor personal, tanto a mí como al equipo nos aportará más conocimientos sobre las tecnologías de *Apple* y dispositivos *iOS*. Dentro de la empresa, ya teníamos una aplicación en *iOS* más enfocada al *self-checkout* y a los clientes de los comercios. No obstante, esta aplicación estaba desactualizada. Por ello, he tenido que investigar el estado del lenguaje de programación actual, así como las arquitecturas utilizadas y el entorno de desarrollo. Esto ha servido también para empezar el camino a una futura actualización de la antigua aplicación con las mismas técnicas aplicadas para este proyecto.

Además de eso, a través de la realización de este proyecto, se han propuesto y aplicado más técnicas de la metodología ágil; y se ha hecho una retrospectiva sobre posibles mejoras de la metodología de trabajo y organización de las tareas. Se han establecido maneras de controlar la calidad del código como revisiones por parte de otros miembros del equipo y la definición de criterios de aceptación.

En cuanto a la vida útil del proyecto, dependerá directamente del modelo de negocio que utilicen los negocios de venta al por menor. Por el momento, este modelo está tendiendo a acercarse a los avances tecnológicos, lo cual nos interesa. Desde *BuyYourself*, se aportan soluciones a varios de los problemas actuales del sector, así que de momento podemos estar tranquilos.

No obstante, si surgiera un modelo alternativo que dejara obsoleto al nuestro, tendríamos o que adaptarlo o que pensar una nueva solución. Sin duda el peor de los casos sería que el comercio en tiendas se eliminara por completo de la sociedad. En ese caso, nuestro producto no tendría validez en el mercado y deberíamos enfocarnos en otro producto. No obstante, este proceso no sucedería de la noche a la mañana, así que nos daría margen para reconducir la empresa.

9. Requisitos

9.1. Historias de usuario

Como ya se ha comentado anteriormente en otros apartados, se separaron las historias de usuario en diferentes bloques de desarrollo para poder estructurar mejor la implementación de cada una de las funcionalidades y planificar mejor los *sprints*. En este apartado, se comentarán en más profundidad estas funcionalidades a través de enumerar las historias de usuario que la comprenden y explicarlas resumidamente. El orden en el que se comentan los bloques y las historias es también el orden en el que se planificó su desarrollo e implementación teniendo en cuenta las dependencias que existen entre ellas.

9.1.1. Login

La funcionalidad de *LogIn* solo cuenta con una historia de usuario, ya que el *logout* se decidió incluir en la parte de ajustes de la aplicación.

- **Como dependiente quiero poder *logearme* (hacer *login*) en la aplicación**: esta historia de usuario comprende las tareas de maquetar la pantalla de *login*; llamar al servidor , a través del servicio correspondiente, una vez se hayan introducido los datos y apretado el botón; y gestionar la respuesta que devuelve el servidor. Si los credenciales son correctos, la sesión del usuario se da por iniciada y se pasa a la pantalla principal de la aplicación para empezar a trabajar. En caso de *login* incorrecto, se informa al usuario de que revise los credenciales y vuelva a probar o que espere para intentarlo de nuevo.

9.1.2. Catálogo y productos

Las historias de usuario relacionadas con este bloque tienen como objetivo que se pueda navegar por el catálogo y sus productos; así como poder ver la información de estos e interaccionar con ellos de forma básica. El catálogo se compone por un número variable de categorías que incluyen un número también variable de categorías. Sigue una estructura arbórea internamente con diferentes niveles. Una vez se llega al nivel hoja, se ve un resumen de los productos (imagen, nombre, código de barras numérico y precio) y se puede hacer *click* en ellos para pasar a la pantalla de detalles de producto.

En la pantalla de detalles del producto se pueden ver los datos básicos de este: nombre, precio, precio anterior sin descuento(opcional), marca, imagen y código de barras (numérico). Además se puede seleccionar la variante del producto que se quiera, en este caso color, para poder consultar su *stock*. Se puede consultar el *stock* tanto de la propia tienda como online y de tiendas cercanas.

- **Como dependiente quiero poder consultar el catálogo de productos**: esta historia de usuario comprende las tareas de llamada al servidor, a través del servicio correspondiente, para recuperar el catálogo; maquetación de la pantalla de catálogo así como la de resumen de productos de cada una; y la navegación por los diferentes niveles de categorías. En caso de no recuperar el catálogo correctamente, se avisa al usuario de que algo ha salido mal y que lo intente más tarde. A posteriori y en rondas de revisión, se añadió la opción de refrescar deslizando el dedo de arriba a abajo en la parte superior del catálogo.

- **Como dependiente quiero poder consultar los datos del producto (variantes, precio, foto, título, marca, barcode):** en esta historia de usuario entrarían las tareas de llamar al servidor, a través del servicio correspondiente, para recuperar la información del producto; y la de maquetar esta información en la pantalla de detalles del producto.

- **Como dependiente quiero poder seleccionar una variante y ver el stock en mi tienda, online y otras tiendas:** esta historia de usuario empieza con la maquetación e implementación de un componente para seleccionar la variante, en este caso el color, del producto. También incluye el maquetado de las tres pestañas de información de *stock* del producto:
    - ***Stock* online:** esta pestaña muestra si el producto con esa variante está disponible online o no.
    - ***Stock* en mi tienda:** esta pestaña muestra cuantos productos disponibles con esa variante hay en la tienda.
    - ***Stock* en otras tiendas cercanas:** esta pestaña muestra cuantos productos disponibles con esa variante hay disponibles en las cinco tiendas más cercanas.

  Una vez se selecciona la variante, se busca en los datos del producto ya cargados de la anterior llamada al servidor para buscar la información del stock para cada una de las pestañas.

9.1.3. Reservas

Este bloque de historias de usuario se centra en las reservas. El sistema permite reservar productos en otras tiendas siempre y cuando haya *stock* en ellas. Para realizar una reserva, primero hay que seleccionar una variante en el apartado de información del producto. Después, si hay stock en alguna de las tiendas cercanas, se puede hacer la reserva apretando el botón correspondiente. La reserva se realiza aportando datos del cliente a través de un formulario. Los campos requeridos son el nombre y apellidos, y o bien el mail o bien el teléfono (o ambos si así lo desea el cliente). Una vez se envía la reserva al servidor, los dependientes de la tienda a la que se ha hecho pueden acceder a su información en el listado de reservas, en la pantalla de pedidos. Los dependientes pueden decidir si aceptar o rechazar la reserva.

- **Como dependiente quiero poder generar una reserva:** esta historia de usuario comprende las tareas de generar interacción con el botón "Reservar" (en la pantalla de detalles de producto, pestaña de Otras Tiendas), mostrar el formulario de datos de la reserva y comunicación con el servidor para generar la reserva, a través del servicio correspondiente. Si en esa tienda no hay productos disponibles, el botón de reserva aparece deshabilitado.

- **Como dependiente quiero poder ver las reservas hechas en mi tienda:** esta historia de usuario comprende las tareas de maquetado de la pantalla de reservas (listado de reservas), recuperar el listado de reservas del servidor (a través del servicio correspondiente) y mostrar estas reservas. Dentro de la tarea de maquetado y recuperación, también entra el organizar las reservas obtenidas por fecha y la capacidad de poder hacer *scroll* hacia abajo y cargar más reservas. A posteriori se añadió un botón de recargar para actualizar el listado de reservas.

- **Como dependiente quiero poder aceptar o rechazar las reservas:** esta historia de usuario comprende las tareas de añadir los botones rechazar/aceptar reserva, realizar la llamada al servidor (a través del servicio correspondiente) para rechazar/aceptar una reserva y gestionar qué pasa con las reservas rechazadas/aceptadas. Una vez una reserva se rechaza, se muestra un mensaje avisando que se ha rechazado con éxito y se elimina de la lista. En caso de ser confirmada, se muestra en el listado como "Confirmada" hasta que se recarga el listado, por si necesita el dependiente esa información para preparar la reserva

9.1.4. Cesta online

Este bloque de historias de usuario se centra en la funcionalidad de la cesta online: una base de datos local que almacena productos. Desde la aplicación, un dependiente puede añadir productos disponibles online (en el *eCommerce*) a la cesta. Una vez añadidos, puede modificar la cantidad de estos en la cesta o eliminarlos. En la cesta se muestran la cantidad y el precio total de los productos. Una vez el cliente quiera finalizar el encargo, el dependiente pasará a la pantalla donde le tomará los datos personales: nombre, dirección, teléfono, mail,... Una vez tomados estos datos procederá a seleccionar el método de envío que el cliente desee de entre los disponibles. En caso de elegir recogida en tienda (siempre disponible), se le mostrará una pantalla con las diferentes tiendas donde desea recogerlo. Una vez confirmados los datos, se pasa al terminal de pago para finalizar la compra. Una vez finalizada, se vacía el carrito y el dependiente puede volver a añadir productos para otro cliente.

- **Como dependiente quiero poder consultar el carrito:** esta historia de usuario comprende las tareas de creación de la base de datos local, maquetación de la pantalla del carrito vacío (cuando no hay productos) y la maquetación del listado cuando hay productos con la información necesaria de cada uno, así como el total de precio y cantidad. Para poder hacer pruebas y ver que se mostraba bien toda la información, se crearon productos de prueba en la base de datos del carrito.

- **Como dependiente quiero poder añadir productos al carrito:** esta historia de usuario comprende las tareas de añadir el botón de "Añadir a la cesta" en la pestaña "*eShop*[9]" de los detalles del producto; mostrar un diálogo de confirmación y añadir el producto a la BD. Una vez el dependiente confirma que quiere añadirlo, se le muestran dos opciones: seguir comprando o ver la cesta. Si desea ver la cesta, se pasa a la pantalla de la cesta con todos los datos actualizados.

- **Como dependiente quiero poder gestionar la cesta:** esta historia de usuario comprende las tareas de añadir los botones que suman/restan cantidad de un producto; el botón de vaciar la cesta y la opción de eliminar un producto (reduciendo su cantidad a 0, cuando aparece un diálogo de confirmación). También entra la tarea de actualizar la BD cuando alguna de estas interacciones se produce.

- **Como dependiente quiero poder rellenar los datos de envío:** esta historia de usuario comprende las tareas de añadir la pantalla de formulario de datos de envío del cliente con los siguientes campos: nombre y apellidos, calle y número, edificio-piso-puerta-etc (opcional), código postal, ciudad, provincia, país, e-mail, confirmación de e-mail y teléfono. Además, incluye las tareas de poder rellenar y comprobar los datos, avisando al dependiente de si falta algún campo o ha sido rellenado de forma incorrecta.

- **Como dependiente quiero poder seleccionar el método de envío:** esta historia de usuario comprende las tareas de recuperar los métodos de envío para los datos dados por el cliente, maquetar la pantalla de selección de datos de envío, maquetar la pantalla de selección de tienda en caso de escoger "Recoger en tienda" y gestionar que solo se pueda seleccionar un solo método de envío.

- **Como dependiente quiero poder finalizar la venta (*redsys*):** esta historia de usuario comprende las tareas de maquetar la pantalla de pago (un componente que muestre una web), recuperar los datos del portal de pago a través de un servicio y conectar con el terminal de pago *Redsys*[10]. Una vez se ha conectado, se debe gestionar la respuesta en función del resultado de la introducción de los datos de la tarjeta del cliente.
    - **Caso error**: si el pago da error, se informa al dependiente y se vuelve a la pantalla de métodos de envío.
    - **Caso ok:** si el pago sucede con normalidad, se avisa al dependiente de que todo ha salido bien, el servidor envía el ticket al cliente por mail, se vacía el carrito y se puede proceder a llenar otra cesta.

[9] **eshop:** tienda online que accede a datos del *eCommerce*
[10] **redsys**: procesador de pago con tarjeta online

9.1.5. Click and collect (Recogida en tienda)

Este bloque de historias se centra en la funcionalidad de *Click&Collect*, es decir, poder pedir los productos desde la web para recoger en la tienda que el cliente desee (siempre y cuando los productos estén disponibles en esa tienda). Una vez un cliente hace un *Click&Collect* en una tienda, a los dependientes de esta les llega la información. Pueden acceder a los *Click&Collects* en la pantalla de pedidos. Una vez acceden a la información, pueden clicar en el *Click&Collect* y acceder a los productos que se han pedido. El dependiente acepta o rechaza cada producto en función de si están disponibles en la tienda. Una vez el dependiente ha interactuado con los productos, envía la confirmación al servidor para cambiar el estado del *Click&Collect*. Si varios o todos los productos han sido aceptados, al dependiente se le muestra la opción de firmar la entrega. cuando el cliente venga a recogerlo, introducirá sus datos y firmará la entrega. Entonces el *Click&Collect* pasa a estar finalizado.

- **Como dependiente quiero poder consultar los *Click&Collect*:** esta historia de usuario comprende las tareas de añadir la pestaña de C&C a la pantalla de pedidos, maquetar el listado de C&C, recuperar el listado de C&C a través del servicio correspondiente, ordenarlos por fecha (al igual que en las reservas) y poder hacer *scroll* para cargar C&C más antiguos. Además se añadió a posteriori, al igual que con las reservas, la opción de recargar el listado.

- **Como dependiente quiero aceptar o rechazar la disponibilidad de los productos del pedido:** esta historia de usuario comprende las tareas de maquetar el listado de productos de cada C&C, maquetar cada producto con sus botones (rechazar/aceptar), enviar la confirmación de disponibilidad a través del servicio correspondiente y tratar la respuesta. Si todos los productos han sido rechazados, el C&C desaparece de la lista una vez se ha hecho la devolución del importe al cliente. Si no han sido todos rechazados, se muestra el botón de firma/recogida en la pantalla de C&C.

- **Como dependiente quiero entregar al cliente su pedido *Click&Collect*:** esta historia de usuario comprende las tareas de mostrar el formulario de recogida. realizar la llamada a través del servicio correspondiente con la información de la recogida y eliminar el C&C una vez entregados los productos disponibles al cliente.

El servidor internamente procesa el estado de los C&C y hace los reintegros de dinero que hagan falta en caso de rechazo de los productos.

9.1.6. Notificaciones

Este bloque de historias de usuario se centra en las notificaciones, es decir, en poder recibir notificaciones en tiempo real de los pedidos para poder tratarlos. No obstante, como se comentará en el apartado de Patrones y Modelo de Datos, así como en el de Planificación Final, hubo unos inconvenientes de peso que hicieron que se tuvieran que dejar estas historias de usuario en estado *blocked* para reanudarlas más adelante.

- **Como dependiente quiero que se muestren las notificaciones:** esta historia de usuario comprende las tareas de configurar el sistema para recibir las notificaciones, maquetar estas notificaciones y mostrarlas al dependiente.

- **Como dependiente quiero que cada vez que reciba una notificación me lleva a la página que hace referencia:** esta historia de usuario se basaba en la tarea de hacer que cuando el dependiente clique en la notificación, la aplicación le muestre la pantalla correspondiente.

9.1.7. Ajustes

Este bloque de historias de usuario se centra en la pantalla de ajustes y sus opciones. Inicialmente se habían planteado el bloque de notificaciones y este como uno solo, pero en vista de los inconvenientes que tuvimos con las notificaciones, se decidió separarlo. Entre las opciones que se incluyen en la primera versión están la de saber en qué tienda estás, con qué usuario y poder cerrar sesión; así como conocer la versión y número de compilación de la aplicación, por si se tuviera que avisar al servicio técnico para así actuar más rápido.

- **Como dependiente quiero poder cerrar sesión:** esta historia de usuario comprende las tareas de maquetar la pantalla de ajustes, donde se muestra el nombre de la tienda actual y las opciones. También maquetar la opción "Cuenta" a través de la cual se accede a la pantalla que muestra el usuario actual y el botón de "Cerrar sesión". Cuando el usuario cierra sesión, se le devuelve a la pantalla de login.

- **Como dependiente quiero poder saber la versión y el número de compilación:** esta historia de usuario comprende las tareas de maquetar la opción "Sistema" de los ajustes, donde aparecen número de versión y de compilación.

9.2. Requisitos no funcionales

Para empezar a comentar los requisitos no funcionales, vamos a empezar comentando lo más importante: el dispositivo donde se ejecutará la aplicación. Como ya se ha comentado en anteriores apartados, la aplicación de dependiente de nuestro sistema está disponible actualmente para *tablets Android*. Esta aplicación en este formato fue pedida por un cliente que la está utilizando actualmente en sus tiendas. Cuando el estudiante se reunió con el equipo *BuyYourself* para comentar la posibilidad de realizar el trabajo final de grado junto a las prácticas, se comentó la posibilidad de llevar a cabo el desarrollo de una aplicación similar para dispositivos *iOS*. Concretamente móviles *iPhone,* para así poder tener más compatibilidad para los clientes que quieran utilizarla. Otro punto fuerte para decidir desarrollarla en *iOS* ha sido la oportunidad de investigar, indagar y aprender sobre tecnologías *Apple*.

En un principio se planteó que la *app* estaría disponible para dispositivos *iPhone* 7 o superior debido a la versión mínima de *iOS* necesaria para ejecutar la aplicación(tal como se comenta en el apartado de la Dimensión Económica). No obstante, a posteriori se investigó más sobre la como poder adaptar la aplicación para más dispositivos. Es por eso que se decidió bajar la versión mínima a la 10.X. Uno de los artículos consultados[17] hace un análisis sobre el tanto por ciento de usuarios de *iPhone* que utiliza cada versión de *iOS* y cada dispositivo. Llegamos a la conclusión que bajando a la 10.X abarcaríamos más mercado. Además, esto nos permitía utilizar el dispositivo de pruebas que tenemos en la empresa (un *iPhone* 5c de segunda mano) y no tener que buscar otro dispositivo más moderno, con la inversión que esto comporta.

Otro requisito funcional para utilizar la aplicación es el acceso a internet. La aplicación necesita poder acceder a internet para ejecutar las funciones básicas y comunicarse con el servidor. Además, se tiene que disponer de la base de datos propia de cada organización cliente en el servidor para acceder a sus datos. Para este proyecto, se utiliza la base de datos de prueba que utilizamos para probar los demás componentes.

10. Arquitectura y estructura del sistema

En este apartado se hará un análisis y comentario de la estructura del proyecto y su arquitectura. Se argumentará el por qué de las elecciones tomadas y se explicarán también las herramientas y componentes que componen el sistema.

10.1. Arquitectura física

En la figura inferior podemos ver los principales elementos que intervienen e interaccionan con la aplicación. Todos estos sistemas a excepción de la base de datos local, también interaccionan con el resto de módulos de la aplicación.

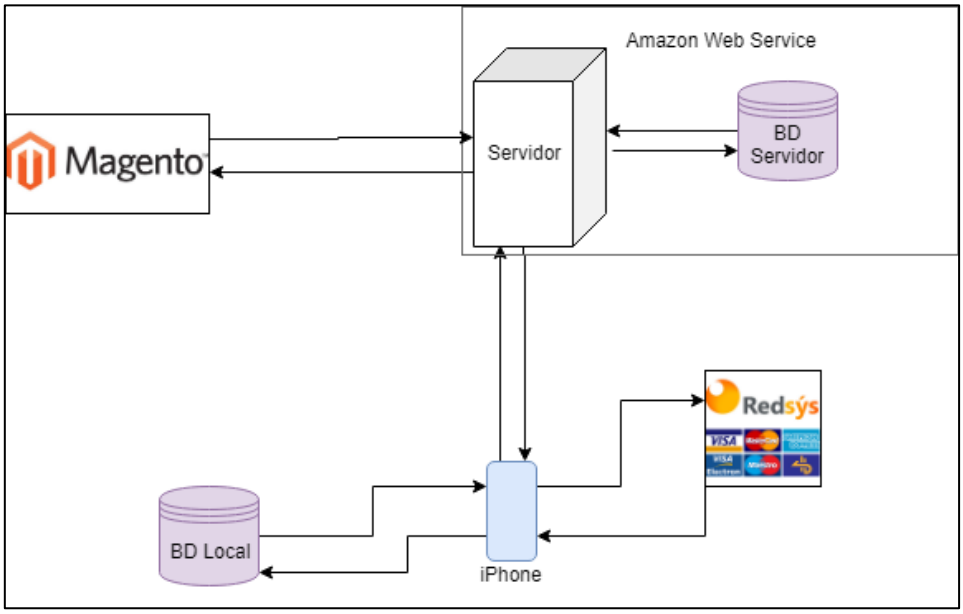


Figura 3: Visión general de la arquitectura física

Primero de todo tendríamos la parte del servidor de *Amazon Web Service*. Aquí están alojados el servidor y la base de datos del servidor. Estos se encargan de procesar y almacenar todos los elementos de las colecciones, así como responder a las peticiones de la aplicación.

Además devuelven los credenciales de *Redsys*, la plataforma de pago online con tarjeta, para poder realizar pagos desde la aplicación. Una vez llegan a la aplicación, esta contacta con *Redsys* para abrir el terminal de venta e introducir los datos de la tarjeta. Una vez introducidos, *Redsys* procesa la respuesta y la aplicación la gestiona.

Por otro lado tenemos el gestor de *Magento*. plataforma de comercio digital que contiene datos de todos los productos, los catálogos, las ventas, los pedidos, etc... A través de sincronizarse con *Magento*, el servidor y su BD consiguen tener la base de datos siempre actualizada con los .

Por último, tenemos la base de datos local, que gestiona la cesta online de cada usuario de la aplicación. Esta base de datos comprende todos los datos de los productos de la cesta para que la aplicación pueda realizar ventas.

10.1.1. Herramientas utilizadas

Para el desarrollo de esta aplicación se han utilizado varias herramientas. Algunas de ellas ya han sido comentadas como *BitBucket*, *Sketch* o *Jira*. Pero aparte de estas se han utilizado algunas que vale la pena comentar:

- ***SmartGit***: plataforma que permite gestionar un repositorio, crear ramas, traerlas a la rama principal, subir código, etc. Todo de manera sencilla y cómoda, con opciones para resolver fácilmente conflictos.
- ***Avocode:*** cuando dejó de estar disponible el ordenador *MAC*, se encontraron ciertas dificultades para poder utilizar la plataforma *Sketch* (nativa de *Apple*) desde la máquina virtual. Entonces se buscó una alternativa y encontramos *Avocode*, una plataforma que permitía visualizar y editar diseños de *Sketch* y tenía todas sus funcionalidades.
- ***Robo3T:*** programa para ver y editar bases de datos *MongoDB*. Se ha utilizado para comprobar los datos del servidor de desarrollo, modificarlos cuando era necesario o ver cómo estaban organizados los datos.
- ***Oracle VM VirtualBox***: herramienta de virtualización a través de la cual se ha virtualizado el sistema *MAC* donde se ha desarrollado gran parte del proyecto.

10.1.2. Tecnologías empleadas

Para el desarrollo del proyecto, hay varias tecnologías implicadas. La primera de todas es *Swift*. *Swift* es el lenguaje de programación de *iOS* que se ha empleado para el proyecto. Concretamente *Swift* 4.2, el que trae la versión de *XCode* utilizada. Se decidió utilizar *Swift* en vez de *Objective-c* (otro lenguaje que se utiliza en desarrollo en *iOS*) porque se asemeja más a *Kotlin*, que es el lenguaje en el que está gran parte de la aplicación de *Android*, y que la adaptación así sería mejor.

Otra tecnología muy importante es *Amazon Web Service*, que aunque no se ha utilizado de forma directa, al ser el sistema del servidor al que se acceden los datos, es relevante comentarla. El servidor está alojado en *Amazon Web Service* y allí tenemos conexiones para las diferentes aplicaciones. Además contamos con un servidor de desarrollo y uno de producción, para no tener problemas con los datos haciendo pruebas ni cargarnos nada.

Para acabar, es necesario recalcar la tecnología *Cocoa Pods*, el sistema a través del cual se han ido añadiendo las diferentes librerías utilizadas, como *Alamofire* para realizar llamadas a servidor o *ObjectMapper* para mapear los objetos. En el archivo *Podfile* se constatan las librerías que se desean instalar en el proyecto y la versión de estas que se desea. Una vez se añaden las líneas correspondientes al *Podfile* para las librerías y versiones que se desean, se ejecutan unos comandos a través de la consola para instalar las librerías. Estos son: "*$ pod install*" para instalar los paquetes de la librería a través de su repositorio, y "*$ pod update*" para actualizar las librerías. Esto permite poder añadir de manera sencilla nuevas herramientas para poder programar las funcionalidades esperadas.

10.2. Arquitectura del sistema

Como se ha comentado previamente, este sistema sigue el patrón arquitectónico Modelo Vista Vista Modelo (MVVM). Tenemos una capa Vista que contiene los archivos referentes a la interfaz gráfica; una capa Vista Modelo que contiene los *ViewModel* anteriormente comentados; y una capa Modelo donde se encuentran los archivos referentes a los datos y los servicios que se comunican con el servidor.

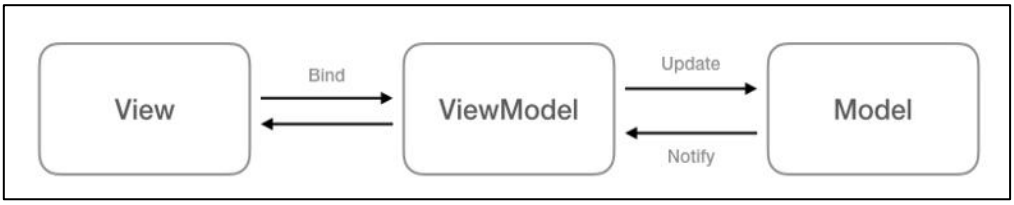


Figura 4: Esquema del patrón MVVM

Para decidir qué patrón se iba a utilizar, se llevó a cabo un poco de investigación que desembocó en un análisis por parte del equipo. En *iOS*, el modelo más utilizado es el MVC (Modelo Vista Controlador). Este modelo en *Swift* no separa entre las funciones de la capa Vista y la capa Vista Modelo anteriormente comentadas. De esta manera, obtenemos clases *ViewConntroller* (Vista + Controlador) que combinan muchos métodos gráficos con los métodos de acceso a la capa de datos (Modelo). Esto provoca una dificultad notable para futuros refactorizaciones a la vez que para poder reusar el código de acceso a la capa de datos en otras funcionalidades similares. Además que carga mucha información a la que la capa visual no necesitaría, información que se debe actualizar manualmente en la vista. En cambio, esta información en el MVVM se actualiza en la vista una vez registra un cambio del *ViewModel.*

Otra razón por la que se utiliza MVVM en este proyecto es porque al separar las funciones importantes de trata de datos en el *ViewModel*, es mucho más fácil poder testearlos independientemente de los métodos gráficos.

El MVVM no es tan comúnmente utilizado en *iOS*, pero es bastante más popular en *Android*. Cuando se comenzó la planificación para implementar la aplicación en i*OS*, se debatió sobre si se podía replicar el MVVM que se utiliza en *Android* en el nuevo proyecto. A través del análisis anteriormente comentado y documentación sobre cómo hacerlo, se decidió aplicarlo.

10.3. Patrones de diseño

En este proyecto se han utilizado algunos patrones de diseño aprendidos en asignaturas de la carrera. Por un lado tenemos el más importante: el patrón *singleton*. Este patrón consiste en crear instancias únicas de una clase para poder utilizar esos datos y funciones en diferentes partes del código tratándolas como un objeto. Este patrón se ha utilizado para implementar los *ViewModel*, así como los servicios.

Al crear un *ViewModel* como *singleton*, se tiene la capacidad de acceder a los datos necesarios a través de diferentes partes del código. Por ejemplo, en el caso de las reservas, querremos acceder desde varias pantallas (productos, para generar reservas, y la pantalla de reservas). Por esto nos es más fácil tener una instancia única con la información y métodos necesarios para hacer gestión de los procesos pertinentes. Además esto nos asegura que los datos se mantengan tal y como se desea en cada momento de la ejecución de la aplicación. Los datos de los *ViewModel* se pueden ir modificando de tal manera que todas las partes de la aplicación que accedan a ella puedan ver estos datos igual.

Otro patrón empleado es el patrón plantilla. Este patrón se utiliza cuando varias entidades comparten parte de sus métodos o estructura, pero también albergan diferencias. Para implementarlo, se crea una superclase abstracta que junta las cosas que tienen en común, y las subclases especializadas extienden de esta. En este proyecto, se ha utilizado para los pedidos. Los pedidos comprenden la información y métodos comunes de las clases que lo extienden: reservas y *click&collect*.

10.4. Modelo de datos

Para explicar el modelo de datos, hemos de separar entre las dos fuentes principales de datos que tiene la aplicación: el servidor y la base de datos local.

En cuanto al servidor, internamente sigue una estructura en documentos siguiendo el sistema *MongoDB*. Su estructura se compone de diferentes colecciones, dentro de las cuales encontramos documentos para cada objeto. Cada objeto tiene un identificador único y estos identificadores pueden usarse para referenciar objetos de diferentes colecciones entre sí. Por ejemplo para referenciar una tienda dentro de una reserva o un producto dentro de cualquier pedido.

Las principales colecciones a las que se accede son las siguientes:

- **Catálogo**: cada documento de esta colección contiene categorías, sub categorías y productos, así como la organización a la que pertenece.
- **Usuarios**: los datos principales que encontramos en estos documentos son usuario, nombre, correo, contraseña, tienda y rol.
- **Tiendas**: los datos principales que encontramos en estos documentos son el nombre de la tienda. el teléfono, la organización, la dirección, la localización y datos de configuración.
- **Organizaciones:** los datos principales que encontramos en estos documentos son el nif, el código de la organización, y datos de configuración.
- **Marcas**: estos documentos contienen el nombre de la marca y su organización
- **Productos:** los datos principales de estos documentos serían el nombre, la descripción del producto, las *url* para acceder a las imágenes del producto), la marca y sus variantes, con sus precios y códigos de barra, organizadas en *sku*[11].
- ***Stockage***: esta colección tiene documentos por tienda y producto. En ellos se muestra el *stock* disponible de ese producto en esa tienda con las correspondientes variante, así como el *stock* reservado de estos.

[11] **sku**: Stock Keeping Unit, sirve para identificar un tipo específico de un producto

- **Reservas**: los datos principales que encontramos en estos documentos son el origen de la reserva (web, aplicación), la fecha de la reserva, información del producto reservado, el estatus de la reserva, datos del cliente, datos del asistente que ha hecho la reserva, tienda, organización y método de envío.
- **Ventas:** los datos principales que encontramos en estos documentos son los productos de la venta, el estatus, los datos del cliente. los datos del asistente, la fecha de compra, el precio, la tienda, la organización, la dirección de envío, el canal, el método de envío y los datos para realizar el pago.

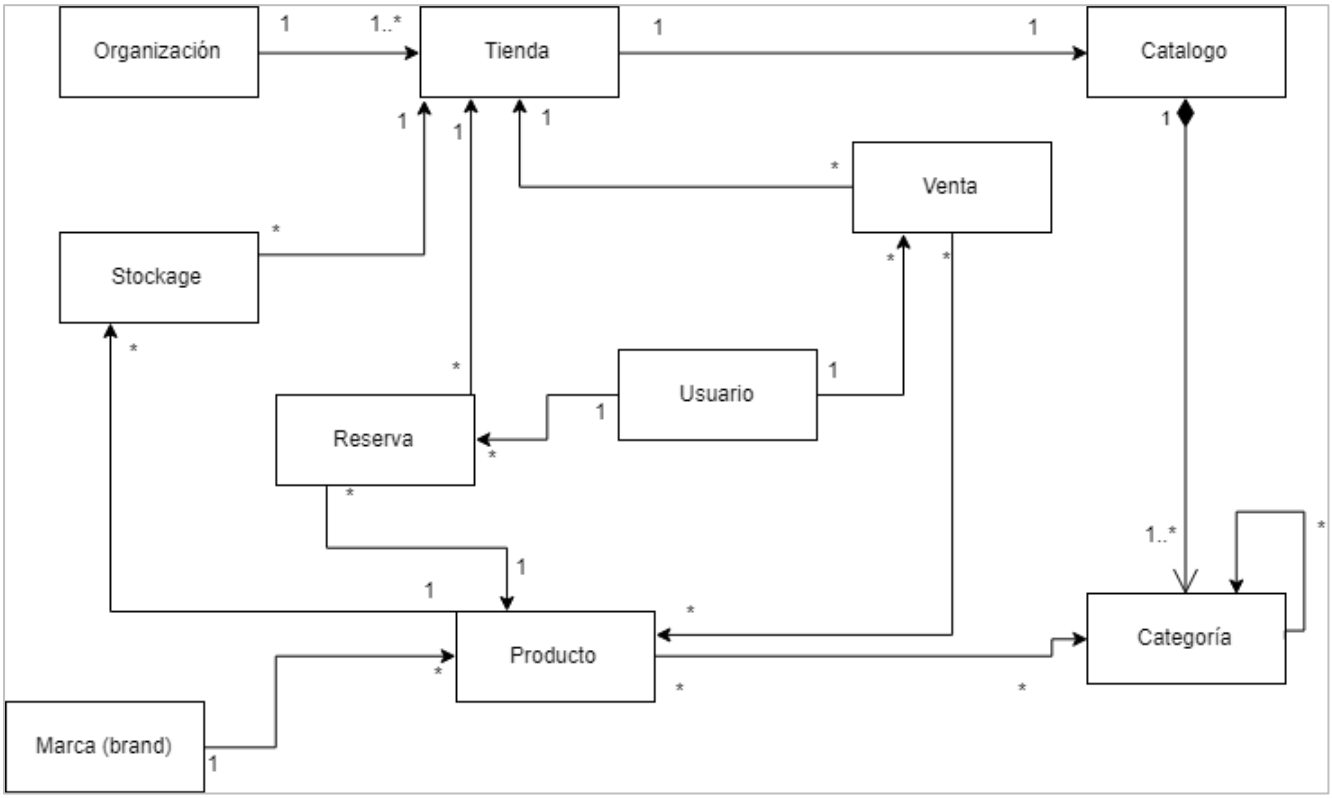


Figura 5: Esquema resumen de la base de datos del servidor

Por otro lado tenemos la base de datos local que gestiona la cesta online. Esta base de datos está diseñada en SQL, utilizando una librería *Sqlite3* para implementarse. Esta base de datos cuenta con una única tabla que relaciona la información necesaria de las diferentes entidades que intervienen. Cada producto de la cesta está formado por información de estas entidades:

- **Producto**: id del producto, nombre y *url* de la imagen.
- ***Sku***: variante, código de barras, precio original y precio actual.
- **Usuario**: nombre de usuario, para poder identificar los productos que añade cada asistente.
- **Organización**: el id de la organización.
- **Tienda**: el id de la tienda.
- **Marca (brand)**: el id de la marca

Además cada fila de la tabla (correspondiente a cada producto) contiene un id propio como clave primaria y un campo cantidad que muestra la cantidad de ese producto. Si un usuario desea añadir un producto que ya está en la cesta, el sistema está preparado para añadir cantidad en vez de duplicar el producto en la base de datos.

La línea de creación de la tabla en *SQL* sería la siguiente:

```
CREATE TABLE "CurrentBasket" (
            id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
            username TEXT NOT NULL,
            productId TEXT NOT NULL,
            name TEXT NOT NULL,
            imageSource TEXT,
            quantity INTEGER NOT NULL,
            originalPrice REAL,
            currentPrice REAL NOT NULL,
            barcode TEXT NOT NULL,
            brand TEXT,
            shopId TEXT NOT NULL,
            organizationId TEXT NOT NULL,
            sku TEXT NOT NULL,
            variant1 TEXT,)
```

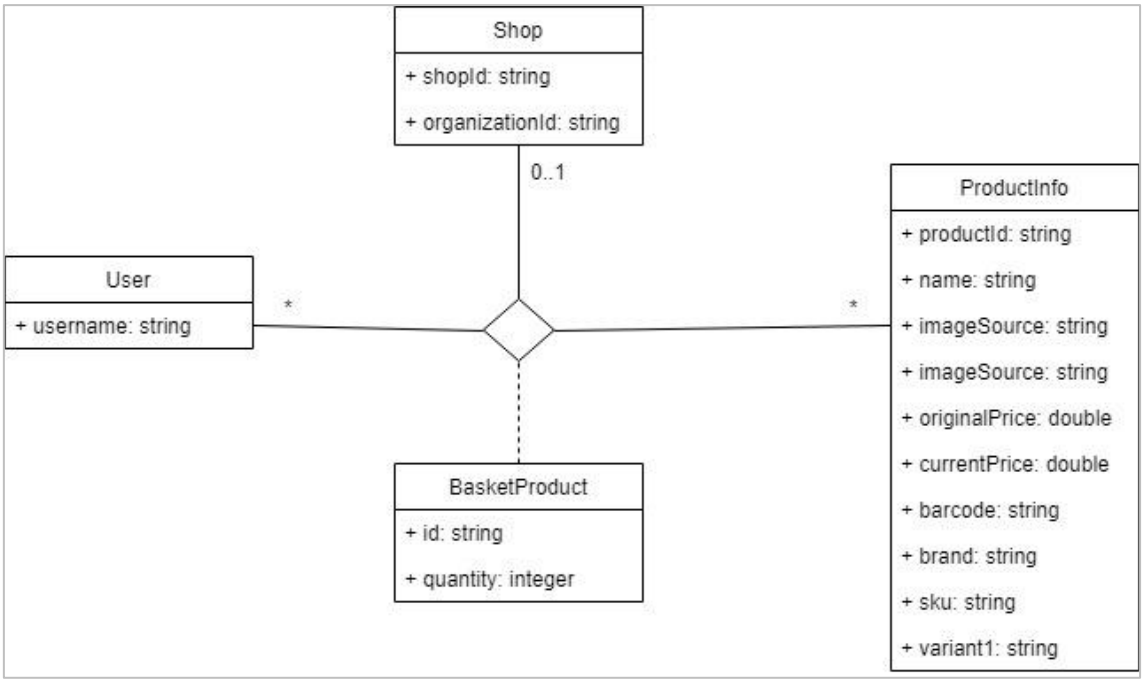


Figura 6: UML de la base de datos local

Otro aspecto importante a destacar sobre el modelo de datos es la utilización de *Firebase*, el gestor de bases de datos de *Google*, para las notificaciones. Así como en la aplicación de *Android*, se pretendía utilizar conexión con *Firebase Cloud Messaging* (el servicio de mensajería a través de la nube de *Firebase*) para recibir las notificaciones. Las notificaciones son generadas por el servidor una vez este detecta que se ha finalizado con éxito la generación de una reserva o un *Click&Collect.* La idea básica de las notificaciones de *Firebase Cloud Messaging* que nosotros utilizamos es que si suscribes varias aplicaciones diferentes a un grupo de notificaciones, estas las podrán recibir de manera sincronizada. No obstante, nos encontramos con un problema para implementar este funcionamiento en *iOS*.

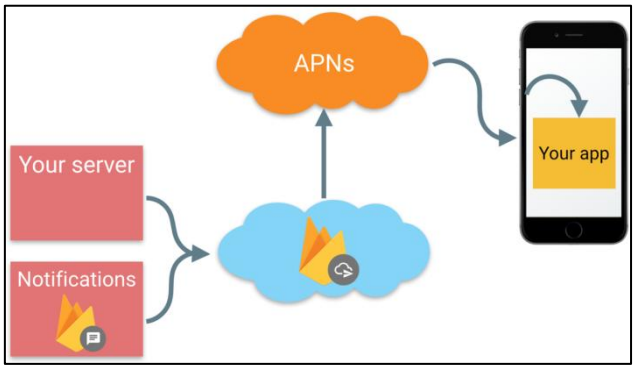


Figura 7: Funcionamiento de *Firebase Messaging* con *iOS*

Para hacer funcionar estas notificaciones en *iOS*, primero tienen que pasar por el sistema de notificaciones de *Apple*, el *APNS (Apple Notification Service),* tal como se muestra en la anterior figura. Para poder acceder a él, hace falta una clave para configurar la aplicación. Clave que se extrae de cuentas de desarrollador *apple* que tengan la licencia activa. Este requisito es también necesario para habilitar las notificaciones en la aplicación. Este no es el caso de nuestra empresa, así que tuvimos que dejar el bloque de las notificaciones de las notificaciones para las perspectivas de futuro. En el apartado de Planificación Final y desviaciones se explica de manera más extensa esta decisión.

10.5. Servicios

Para empezar a comentar el apartado de servicios, se ha de explicar el funcionamiento de los *Managers.* Los managers son archivos que contienen las funciones básicas para realizar las conexiones con las bases de datos.

Los primeros *Managers* importantes son el *AccessTokenManager* y el *SessionManager*. Estos se encargan de establecer una sesión de conexión con el servidor y mantenerla. El *AccessTokenManager* crea un *Token* de acceso que, mientras no esté caducado, permite hacer las llamadas al servidor. El *SessionManager* se encarga también de los procesos internos de *login* y *logout*, manejando los credenciales necesarios para iniciar sesión y cerrarla.

A continuación se encuentra el *ConnectionManager*, que es el principal encargado de crear conexiones con el servidor, identificar los lugares del servidor de donde se debe extraer la información y realizar las llamadas correspondientes. Este manager contiene toda la información de las *url* de acceso a datos y es capaz de codificar los parámetros necesarios para realizar las llamadas. Desde los servicios, se llama a este manager para realizar cualquier llamada GET,POST,PUT contra el servidor con los datos correspondientes.

Además, realiza comprobaciones llamando al *AccessTokenManager* y al *UserService*, el cual se comentará más tarde, para asegurarse de que los credenciales del usuario permiten realizar la llamada y que el *Token* de acceso no haya caducado. Si estas condiciones no se cumplen, se devuelve al usuario a la pantalla de *login*.

Por último la aplicación también cuenta con un *OnlineBasketDBManager*, el cual se encarga de todas las gestiones de la base de datos local de la cesta online. Cuando se inicia la aplicación, se crea la base de datos *SQL* si no existe. Después, será el encargado de añadir los productos con sus campos correspondientes, aumentar o disminuir las cantidades, eliminar productos, y realizar las consultas necesarias para mostrar los productos y poder finalizar la venta.

Ya pasando a los servicios, la aplicación cuenta con los siguientes:

- ***UserService:*** este servicio se encarga de extraer la información del usuario al hacer *login*, actualizarla si hiciera falta y de cerrar la sesión.
- ***CatalogService***: este servicio se encarga de realizar la llamada que recupera el catálogo del servidor.
- ***ProductService***: este servicio se encarga de realizar la llamada para recuperar la información de los detalles del producto.

- ***PreOrderService***: este servicio se encarga de realizar la llamada correspondiente a la creación de reservas, recuperar las reservas de la tienda actual, y realizar la llamada de acepto/rechazo de estas.
- ***DeliveryMethodService***: este servicio se encarga de realizar la llamada correspondiente para recuperar los métodos de envío disponibles del servidor para una venta en concreto.
- ***PaymentGatewayService:*** este servicio se encarga de realizar la llamada que recupera los credenciales necesarios para cerrar la venta a través de *Redsys*.
- ***ClickAndCollectService***: este servicio se encarga de realizar las llamadas de recuperación de los *Click&Collects*, así como las de envío de confirmación de disponibilidad y entrega de los productos al cliente.

11. Implementación

11.1. Estructura del proyecto

El proyecto ha sido desarrollado en *XCode* 10, la plataforma de desarrollo de aplicaciones para sistemas *iOS* de *Apple*. Aquí se creó un proyecto ordenado por carpetas para clasificar los diferentes archivos que componen la aplicación. Este proyecto ha sido organizado según el patrón arquitectónico *Model-View-ViewModel*, como ya se comenta en el apartado de Arquitectura y Estructura.

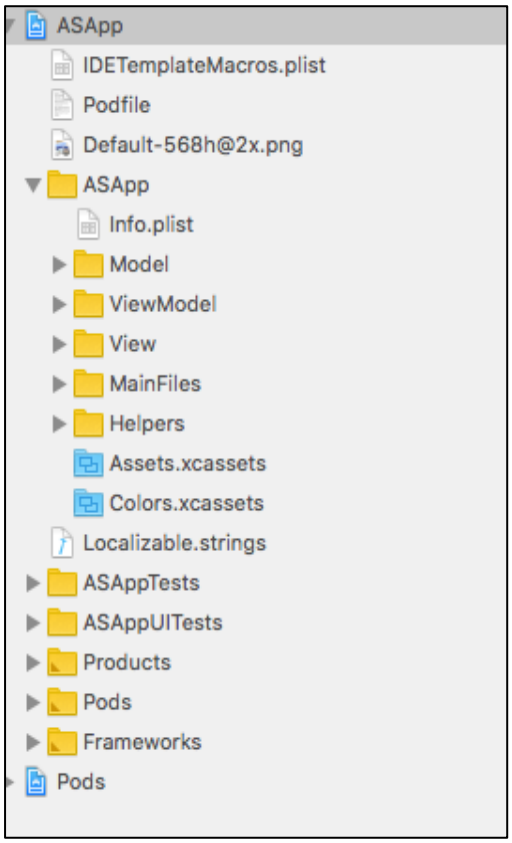


Figura 8: Estructura del proyecto en *XCode*

Una capa *View* que alberga todo lo referente a la capa del *front-end*, la capa visual que ve el usuario con todos sus componentes. Aquí encontramos la vista de cada pantalla. así como las de las celdas de los listados y colecciones, que tienen sus propias funciones.

Una capa *ViewModel*, que comprende todos los *ViewModel*, es decir, los archivos donde se encuentran todas las funciones que comunican la *View* y la capa *Model*.

Una capa *Model* que comprende todos los modelos de datos y clases utilizados, así como los servicios y *managers*(archivos que contienen las funciones más importantes para comunicar con el servidor, a través de las cuales hacen la llamada los servicios) empleados para comunicarse con el servidor; así como todo lo referente a gestión de la base de datos local.

Además de las carpetas de las diferentes capas, tenemos otras carpetas como la carpeta *MainFiles*, donde se encuentran los archivos principales de los proyectos *XCode*: *MainStoryboard, LaunchScreenStoryboard* y el *App Delegate.* Estos son los archivos que gestionan el flujo general de la aplicación. Concretamente el primero es el archivo donde se maquetan las pantallas y se establece el flujo de pantallas.

También tenemos la carpeta *Helpers*, donde se encuentran los archivos de ayuda con funciones o extensiones que se utilizan en toda la aplicación.

En los *Assets* tenemos todas las imágenes e iconos utilizadas y en los *Colors* los colores empleados.

Otros archivos que está bien comentar son el *Info.plist*, que es el archivo de configuración de la aplicación; el archivo *Localizable.Strings,* donde se encuentran organizados todos los textos de formato *string* que se han utilizado en la aplicación. Esto permite que cuando queramos añadir otro idioma a la aplicación, simplemente tengamos que traducir este archivo y no duplicar código para sustituir esas líneas de *string*.

Por otro lado tenemos el archivo *Podfile*, que se utiliza para actualizar e instalar las librerías necesarias para el desarrollo y funcionamiento de la aplicación. Sobre esto se hablará más en profundidad en el siguiente sub-apartado.

11.2. Llamadas al servidor

En esta parte del apartado de implementación, se comentará un poco más en profundidad la implementación y el funcionamiento de los servicios, citando las funciones principales y ejemplificando.

Para realizar las conexiones con el servidor, se utiliza una librería llamada *Alamofire*, que permite hacer llamadas HTTP [12] en lenguaje *Swif.* Esta librería ha sido instalada a través de *CocoaPods* como todas las demás. A continuación se expone un ejemplo: la llamada GET del *ConnectionManager*, a través de la cual se hacen este tipo de peticiones.

```
func requestGetConnection<T: BaseMappable>(
isUserAuthenticatedRequest:Bool, url: String, responseSuccess:
@escaping (T) -> Void, responseError:  @escaping (Error?, Data?) ->
Void){

        return self.requestConnection(isUserAuthenticatedRequest:
isUserAuthenticatedRequest, url: url, httpMethod: .get, body: nil,
responseSuccess: responseSuccess, responseError: responseError)

}
```

Como se puede observar, esta llamada trata la recuperación de objetos *BaseMapable*. Estos objetos son tales que pueden construirse a través de un mapa de datos extraído del servidor. Para poder tratar con estos objetos, se instaló la librería *ObjectMapper*. Esta librería permite definir objetos como "*Mappable*".

[12] **HTTP**: abreviatura de la forma inglesa *Hypertext Transfer Protocol*, 'protocolo de transferencia de hipertextos', que se utiliza en algunas direcciones de internet.

Para que un objeto pueda ser *Mappable*, ha de cumplir un protocolo, es decir, contar con sus funciones básicas. Estas son la función `init?(map)` y la función `mapping(map)`. Estos permiten inicializar un objeto a través de un mapa y mapear los campos de manera correspondiente como se observa en el siguiente ejemplo:, que corresponde a la respuesta de la llamada de *login (LoginResponseDto).*

```
// Initializers
required init?(map: Map) {

}

// Mappable
func mapping(map: Map) {
    access_token    <- map["access_token"]
    token_type      <- map["token_type"]
    refresh_token   <- map["refresh_token"]
    expires_in      <- map["expires_in"]
    scope           <- map["scope"]
    error           <- map["error"]
    errorDescription <- map["error_description"]
}
```

El mapa o diccionario está compuesto por claves y valores. Por lo tanto, para realizar el mapeo correctamente, se tiene que asignar cada campo con el valor dentro del mapa al que corresponde su nombre. De esta manera se logra traspasar de los datos del servidor, que están en formato mapa o diccionario, a la estructura de datos deseada en la aplicación para tratarlos.

11.3. Patrón *singleton*

Como se ha comentado en previos apartados, los servicios y los Vista Modelo están basados en un patrón *singleton*. Para implementar esto, se ha declarado una propia instancia "*shared*" dentro del propio archivo a través de la cual acceden los demás. A continuación se presenta un ejemplo con la línea que encabeza el *CatalogViewModel.swift*:

```
static let shared = ProductsViewModel()
```

A partir de esta línea, se crea el *ViewModel* como instancia única y estática. Para acceder a él por ejemplo desde la pantalla de información del producto, se declara una variable que sea igual a esta variable *shared*; y dentro del código se llama a esta variable para acceder a los métodos. Aquí unas líneas de la vista de detalle de producto que sirven de ejemplo para lo mencionado:

```
var productViewModel = ProductsViewModel.shared
let variant = productViewModel.getCurrentVariant()
```

11.4. Observables

Para la implementación de esta aplicación se han seguido los principios de la Programación Reactiva. Esto significa que la aplicación está diseñada para reaccionar a los cambios y actualizar los datos. Esto es especialmente importante ya que las llamadas al servidor son asíncronas, es decir, no se ejecutan en la misma secuencia que el resto del código. Cuando se lanza una llamada al servidor, el código no se queda esperando la respuesta.

Es por esto que se necesita de un mecanismo dentro de la aplicación para actualizarse una vez llegan las respuestas del servidor. Aquí entra el uso de los observables. El funcionamiento básico de estos objetos es el siguiente: desde una clase se crea una variable y se le define un observable para su valor. Después desde la clase que se quiera observar este objeto, se crea una suscripción a él. Esta suscripción puede actuar en varios momentos. En este caso, las suscripciones actúan cuando detectan un evento "onNext", que viene a ser cuando cambia de valor el objeto. Aquí se puede definir lo que debe hacer el programa una vez detecte el evento. A continuación tenemos el ejemplo del Catálogo.

En el *CatalogViewModel*, se definen la variable y su observable:

```
private var catalogVariable = Variable(CatalogDto())
    var catalogObserver:  Observable<CatalogDto?> {
        return catalogVariable.asObservable()
    }
```

Desde la parte de la vista del catálogo, se crea la suscripción y se define el comportamiento, que será el de actualizar el catálogo con los nuevos valores dentro de la tabla que se muestra.

```
  catalogViewModel.catalogObserver.subscribe(onNext: { [weak self]
                  catalog in
                self?.catalogCategories = catalog?.categories
                self?.dataSource.reloadData()
            }).disposed(by: disposeBag)
```

Para que el observable no siga cargando información o perjudicando el rendimiento, se utiliza la función `disposed(by: disposeBag)` para que libere su memoria hasta que vuelva a ser necesitado.

Estos observables y todas sus funciones se encuentran en la librería *Reactive Swift (RxSwift).*

12. Pruebas funcionales

A lo largo del desarrollo de la aplicación se han llevado a cabo pruebas funcionales sobre cada una de las funcionalidades que se iban añadiendo. Estas pruebas han sido realizadas en el *iPhone 5c* del que se dispone en la oficina. Cada vez que se realiza un *pull-request* (tal como se explica en el apartado de Metodología), se pasa primero una revisión del código para detectar que el código esté limpio, que sigua las normas de estilo establecidas, que guarde coherencia y que no repita código. También se revisa que esté todo bien documentado y sea comprensible para cualquier desarrollador que quiera indagar en la implementación de la aplicación.

El revisor de la *pull-request* puede rechazarla si no cumple con los estándares mínimos o si tiene demasiados errores. En este caso, incluirá los comentarios necesarios en el mensaje de rechazo para que el desarrollador pueda hacer los cambios pertinentes. una vez realizados, se crea una *pull-request* nueva para pasar otra vez la revisión. Si son fallos leves, el revisor avisará al desarrollador para que realice los cambios y actualice la *pull-request*, para así ser aceptada. El revisor puede añadir también comentarios al mensaje de acepto para tomar anotaciones, por si en un futuro se puede estudiar mejorar algunos aspectos.

Una vez aceptada la *pull-request*, esta se integra en la rama principal de desarrollo. Cuando se han solucionado los posibles conflictos de código entre las dos ramas, comienza el periodo de pruebas. En este periodo, se llevan a cabo pruebas funcionales entre el desarrollador y el director del proyecto que hace de revisor. Se prueba que la aplicación, una vez aplicados los cambios, cumpla todos los requisitos previos que se estipularon en el *Backlog Refinement*.

Las pruebas que se realizan son manuales siguiendo la interacción esperada. En ocasiones se compara esta interacción y los tiempos de respuesta con la versión de *Android*. A medida que se realizan estas pruebas, se documentan los posibles *bugs* gráficos o las posibles mejoras de rendimiento que se pueden realizar. Para esto tenemos una hoja de cálculo en *Google Drive* con el nombre del *bug* o mejora, una breve descripción y la funcionalidad o sección de código a la que atañe.

A través de lo escrito en esta hoja de cálculo, la aplicación ha ido sufriendo cambios. Ya sea justo después de realizar una revisión o después de acabar un bloque de historias de usuario, se le ha dedicado un tiempo a revisar las mejoras, priorizarlas y solucionar las máximas posibles.

Una vez acabó el desarrollo de las funcionalidades principales, se llevó a cabo una revisión general de toda la aplicación así como de la lista de mejoras pendientes. De este modo, se comenzó el periodo de modificaciones finales para mirar de aportar el máximo valor posible a la aplicación.

Además, se les dejó la aplicación a los otros dos miembros del equipo que no habían estado tan implicados en el desarrollo, más allá de planificación y las pequeñas demos que se realizaban al cerrar el *sprint*, para que navegaran por la aplicación e intentaran descubrir posibles fallos. Los pequeños fallos encontrados, fueron anotados en la hoja de cálculo, se priorizaron y se trabajó en ellos para mejorar la aplicación.

13. Planificación Final

En general, la planificación inicial se ha seguido bastante bien hasta los últimos *sprints*, en los que algunas tareas se han alargado debido a varias desviaciones. Esto ha provocado que a los últimos bloques de funcionalidades, no se les pudiera dedicar el tiempo esperado en la planificación inicial.

La primera desviación importante fue debida a un cambio de entorno de trabajo. El proyecto se estaba desarrollando en un *MAC* propiedad de un socio de la empresa. Llegado el momento, este socio quiso ponerlo a la venta, así que nos vimos obligados a buscar una alternativa. Tuvimos que dedicar varias jornadas a preparar un entorno de máquina virtual *MAC* para ser ejecutado en *Ubuntu* mediante *Virtual Box*.

No obstante, se hizo presenta la necesidad de adquirir más memoria *RAM* para poder ejecutar la máquina virtual en el ordenador de sobremesa destinado para ello. Fueron surgiendo algunos problemas con la instalación de algunos programas y los drivers, pero fueron más fáciles de solucionar. Además, desde la máquina virtual no conseguimos conectar el *iPhone* de pruebas para instalarlo, así que tuvimos que plantear una alternativa: instalarlo desde el ordenador del director del proyecto para hacer las pruebas y la revisión de funcionalidades.

Esta desviación provocó un retraso de casi un *sprint*, dado que se sumó a problemas para conectar con el servidor. Se tuvieron que hacer modificaciones importantes en el *ConnectionManager* y en algunos servicios en vista de que para recuperar las reservas y los *click&collects*, se necesitaba recuperar colecciones de objetos. Hasta ese entonces, todas las llamadas recuperaban un solo objeto que contenía toda la información, por ejemplo un producto o un catálogo entero como único objeto. Para ello, tuvimos que adaptar las llamadas al servidor para admitir este comportamiento.

Esto provocó que las reservas se alargaran un *sprint* más de la cuenta. Estaba planeado acabarlas entre la última mitad del *sprint* 5 y primera la mitad del *sprint* 6. Finalmente se acabaron entre el *sprint* 6 y el 7, ocupando la totalidad de este último. Por tanto la cesta online, que estaba planificado empezar su desarrollo a mitad del *sprint* 6, comenzó en el *sprint* 8, ocupando la totalidad de los *sprints* 8 y 9. Esto a su vez desplazó el desarrollo de los *click&collects* a los *sprints* 10 y 11.

La cesta no quedó cerrada en el *sprint* 9 debido a otro contratiempo: encontramos un error al intentar volver a acceder a la pantalla de pago cuando los datos de la tarjeta de crédito eran incorrectos o se anulaba la transacción mediante "cancelar" dentro de la pantalla de *redsys.* Cuando desde la aplicación se intentaba volver a acceder a esta pantalla recuperando los credenciales de *redsys*, había veces en que funcionaba con normalidad y otras que no.

Tras detectar que el problema no parecía encontrarse en cómo se enviaban desde la aplicación los datos de venta al servidor para recuperar los credenciales de *redsys*, se decidió dejar este error apartado y proseguir con el desarrollo de los *click&collects*. Una vez acabados, se retomó la búsqueda de soluciones de este error, pero en vista de no encontrar ninguna lógica, se decidió seguir acabando el proyecto y buscar más información para solucionarlo más adelante.

Otra desviación que nos encontramos fue la extensión del *sprint* 11. Este sprint se extendió dos semanas extra para sincronizar al equipo *BuyYourself* con el equipo de Misako, empresa para la que desarrollamos y mantenemos su software en tiendas. Así pues, en vez de finalizar el último *sprint* de desarrollo el 28 de diciembre como se tenía planeado, se acabó el 11 de enero. No obstante, en este *sprint* entraron las tareas que permitían cerrar el bloque *Click&collect*, las historias de usuario del bloque de Ajustes y las primeras pruebas finales y revisiones.

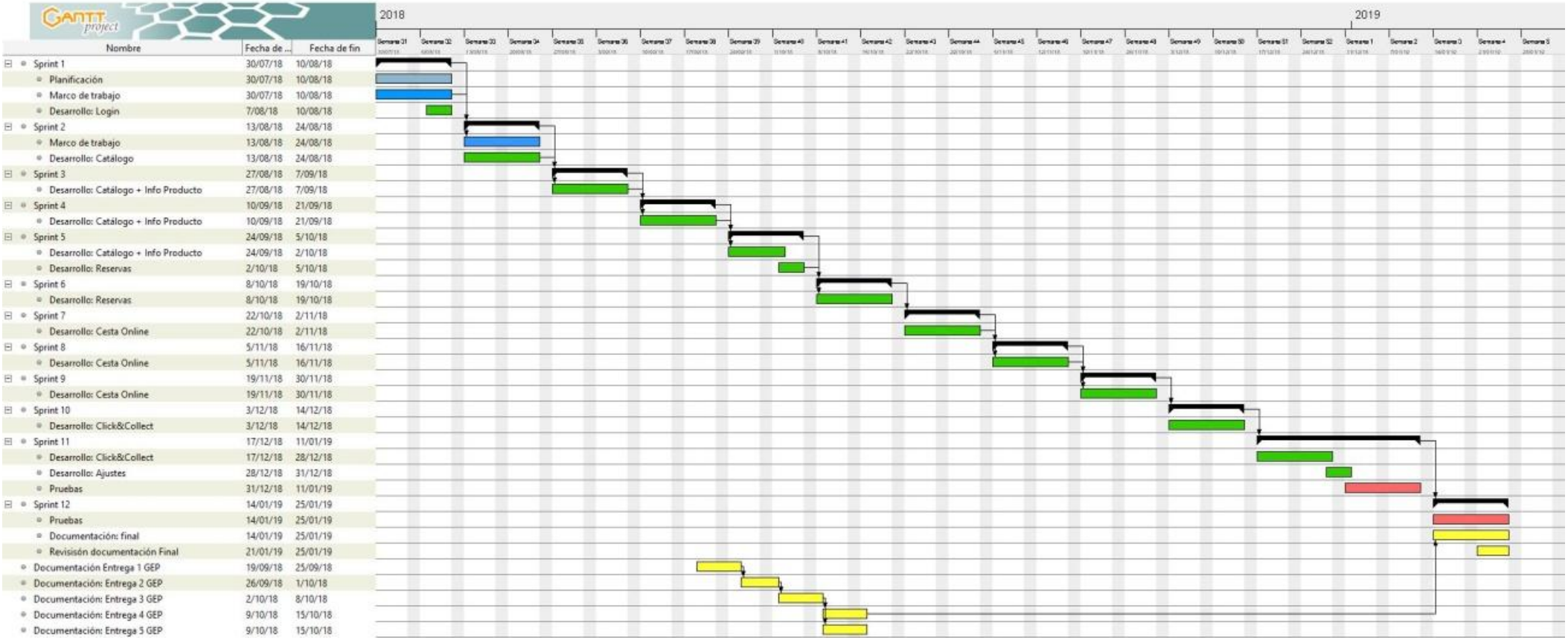
En este último *sprint* de desarrollo también estaban planificadas las historias de usuario del bloque de Notificaciones. Cuando se comenzó a adaptar la aplicación para poder recibir notificaciones de *Firebase*, tal como se explica en apartados anteriores, nos encontramos con un gran problema: se requería una licencia de cuenta de desarrollador *apple* activa para poder habilitar las notificaciones en la aplicación, y descargar una clave a través de la cuenta para habilitar las notificaciones en *Firebase*. Esta licencia cuesta 100€ y hay que renovarla anualmente.

Como actualmente en *BuyYourself* no se tiene ninguna aplicación iOS comercial funcionando, esta licencia no se había renovado. Nos vimos en la encrucijada de decidir si merecía la pena renovar la licencia solo para este proyecto, ya que no iba a salir directamente al mercado una vez finalizado su desarrollo. Finalmente se decidió no renovarla y dejar las notificaciones para perspectivas de mejora o para el momento en el que encontremos un cliente que nos pida esta aplicación concretamente, ya que ahora mismo la empresa no podía asumir este gasto sin rentabilizarlo.

Todas estas desviaciones han repercutido en el total de horas final de desarrollo. No obstante, dentro del presupuesto se habían destinado 1.400€ como contingencia por posibles desviaciones temporales. Estas desviaciones han repercutido sobre todo a las horas que ha dedicado el estudiante (al ser ampliadas las horas totales de desarrollo de algunas funcionalidades y reducidas otras), por lo que ha quedado cubierto por este presupuesto.

Por otro lado, también nos encontramos con una desviación en el plano económico que nos ha resultado favorable. Finalmente no necesitamos adquirir otro dispositivo *iPhone* al bajar la versión mínima y pudimos utilizar el *iPhone 5c* del que disponemos en la oficina y que se utilizó hace unos años para una primera versión de la aplicación de comprador.

13.1. Diagrama de Gantt Final



14. Conclusiones

Una vez finalizado el proyecto, las sensaciones generales que ha dejado el final del proyecto son las de una aplicación usable, cómoda y útil; que podría ponerse a funcionar en una tienda con muy pocas modificaciones. Pese a no haber podido desarrollar todas las funcionalidades esperadas y haber encontrado pequeños *bugs* que no se han podido solucionar, el resultado del proyecto ha sido satisfactorio. Se ha conseguido desarrollar la aplicación sin exceder el presupuesto, y los contratiempos y desviaciones han podido ser mitigados. Al final, se ha extraído mucho conocimiento del desarrollo del proyecto.

Cuando se comenzó a hablar sobre qué se iba a desarrollar, al estudiante se le propuso el reto de aprender a programar una aplicación de inicio a fin en un entorno nuevo: el de desarrollo para *iOS*. Para ello, debía aprender a programar en lenguaje *Swift*. El parecido que *Swift* alberga con *Kotlin* —lenguaje conocido previamente por el estudiante a través de trabajar en la empresa— fue de gran ayuda para empezar. Además, gracias a este proyecto, el estudiante ya ha adquirido suficientes conocimientos como para poder programar aplicaciones en cualquier entorno y poder entrar a formar parte de cualquier equipo de desarrollo.

Por otro lado, el estudiante descubrió una activa comunidad de desarrolladores para dispositivos *iOS* que suben contenido a internet con tutoriales, consejos o explicaciones. Gracias a esta comunidad, fue fácil encontrar opciones sobre componentes y tipos de objetos para usar en la aplicación, así como dónde encontrar las librerías necesarias y su documentación.

Este proyecto ha servido también para aplicar nuevas dinámicas en el equipo y adaptarse mejor a las metodologías ágiles. El equipo ha introducido nuevas reuniones de planificación como el *Backlog Refinement* o las pequeñas reuniones que llevan a cabo estudiante y director antes de desarrollar la aplicación. Además, ha implantado el esquema *GitFlow* con eficacia, lo que ha mejorado la capacidad de prueba y revisión del código.

Después de la presentación de este documento y la defensa delante del tribunal, se prevé comenzar a planear el futuro de la aplicación: se revisarán las historias de usuario que fueron planteadas al inicio —pero que no entraron en el proyecto—; se estudiará a cuales dar prioridad y que funcionalidades se podrían añadir; y comenzará una nueva ronda de desarrollo, si las circunstancias son favorables. En el caso de que una empresa se interesara por la aplicación, se le dedicaría más tiempo y recursos.

En conclusión, la aplicación ha logrado cumplir las expectativas que se tenían de ella pese a los problemas ocurridos durante el desarrollo. Además, llevar a cabo este proyecto ha sido muy positivo y beneficioso tanto para el estudiante como para el equipo.

15. Referencias, webgrafía y bibliografía

[1] Amazon.com. (n.d.). *Amazon.com: : Amazon Go*. [online] Available at: https://www.amazon.com/b?ie=UTF8&node=16008589011 [Accessed 21 Sep. 2018].

[2] La plataforma mobile para retail. (n.d.). *Retail Solutions - La plataforma mobile para retail*. [online] Available at: https://buyyourself.io/ [Accessed 21 Sep. 2018].

[3] Misako.com. (n.d.). *Bolsos, maletas y mochilas | MISAKO Shop Online*. [online] Available at: https://www.misako.com/ [Accessed 22 Sep. 2018].

[4] Epson.es. (2015). *El retail avanza hacia la máxima personalización*. [online] Available at: https://www.epson.es/insights/article/el-retail-avanza-hacia-la-maxima-personalizacion [Accessed 22 Sep. 2018].

[5] Sanahuja, R. (2015). *Los españoles son los compradores que menos aguantan la espera en las tiendas*. [online] Epson.es. Available at: https://www.epson.es/insights/article/los-espanoles-son-los-compradores-que-menos-aguantan-la-espera-en-las-tiendas [Accessed 22 Sep. 2018].

[6] Debitoor.es. (n.d.). *e-commerce - ¿Qué es el e-commerce?*. [online] Available at: https://debitoor.es/glosario/definicion-ecommerce [Accessed 23 Sep. 2018].

[7] MishiPay. (n.d.). *Home - MishiPay*. [online] Available at: https://mishipay.com/ [Accessed 23 Sep. 2018].

[8] NewStore. (n.d.). *Omnichannel Now. Omnichannel Retailing Platform Built Mobile-First.*. [online] Available at: https://www.newstore.com/ [Accessed 23 Sep. 2018].

[9] Proyectos Ágiles. (n.d.). *Qué es SCRUM*. [online] Available at: https://proyectosagiles.org/que-es-scrum/ [Accessed 24 Sep. 2018].

[10] Atlassian. (n.d.). *Jira | Software de seguimiento de proyectos e incidencias | Atlassian*. [online] Available at: https://es.atlassian.com/software/jira [Accessed 24 Sep. 2018].

[11]Bitbucket. (n.d.). *Bitbucket | The Git solution for professional teams*. [online] Available at: https://bitbucket.org/ [Accessed 25 Sep. 2018].

[12] Atlassian. (n.d.). *Gitflow Workflow | Atlassian Git Tutorial*. [online] Available at: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow [Accessed 24 Sep. 2018].

[13] Nextail.co. (n.d.). *Nextail | Dynamic inventory optimization for the next generation of retail*. [online] Available at: http://nextail.co/ [Accessed 25 Sep. 2018].

[14] Mercaux.com. (n.d.). *Mercaux*. [online] Available at: https://mercaux.com/ [Accessed 25 Sep. 2018].

[15] Sketch. (n.d.). *The digital design toolkit*. [online] Available at: https://www.sketchapp.com/ [Accessed 25 Sep. 2018].

[16]Fib.upc.edu. (2018). *Prácticas en empresa | Facultad de Informática de Barcelona*. [online] Available at: https://www.fib.upc.edu/es/empresa/practicas-en-empresa [Accessed 12 Oct. 2018].

[17]Smith, D. (2018). *iOS Version Stats - David Smith, Independent iOS Developer*. [online] David-smith.org. Available at: https://david-smith.org/iosversionstats/ [Accessed 20 Jan. 2019].

[18]Developer.apple.com. (n.d.). *Xcode - Apple Developer*. [online] Available at: https://developer.apple.com/xcode/ [Accessed 20 Jan. 2019].

[19]Avocode. (n.d.). *Hand-off and Inspect Sketch, PSD, XD, AI, & Figma designs | Avocode*. [online] Available at: https://avocode.com/ [Accessed 24 Jan. 2019].

[20]Developer.apple.com. (n.d.). *Apple Developer*. [online] Available at: https://developer.apple.com/ [Accessed 21 Jan. 2019].

[21]Firebase. (n.d.). *Agrega Firebase al proyecto de iOS | Firebase*. [online] Available at: https://firebase.google.com/docs/ios/setup?hl=es-419 [Accessed 24 Jan. 2019].

[22]GitHub. (n.d.). *stephencelis/SQLite.swift*. [online] Available at: https://github.com/stephencelis/SQLite.swift [Accessed 24 Jan. 2019].

[23]Inc., A. (n.d.). *Swift.org*. [online] Swift.org. Available at: https://swift.org/getting-started/ [Accessed 21 Jan. 2019].

[24]Magento. (n.d.). *eCommerce Platforms | Best eCommerce Software for Selling Online | Magento*. [online] Available at: https://magento.com/ [Accessed 24 Jan. 2019].

[25]Medium. (2017). *How not to get desperate with MVVM implementation – Flawless App Stories – Medium*. [online] Available at: https://medium.com/flawless-app-stories/how-to-use-a-model-view-viewmodel-architecture-for-ios-46963c67be1b [Accessed 24 Jan. 2019].

[26]MongoDB. (n.d.). *Open Source Document Database*. [online] Available at: https://www.mongodb.com/es [Accessed 24 Jan. 2019].

[27]Redsys.es. (n.d.). *Redsys | Servicios de procesamiento*. [online] Available at: http://www.redsys.es/ [Accessed 21 Jan. 2019].

[28]Robomongo.org. (n.d.). *Robo 3T - formerly Robomongo — native MongoDB management tool (Admin UI)*. [online] Available at: https://robomongo.org/ [Accessed 24 Jan. 2019].

[29]Sqlite.org. (n.d.). *SQLite Documentation*. [online] Available at: https://www.sqlite.org/docs.html [Accessed 24 Jan. 2019].

[30]Amazon Web Services, Inc. (n.d.). *AWS | Cloud Computing - Servicios de informática en la nube*. [online] Available at: https://aws.amazon.com/es/ [Accessed 24 Jan. 2019].

[31]Team, C. (n.d.). *CocoaPods.org*. [online] Cocoapods.org. Available at: https://cocoapods.org/ [Accessed 24 Jan. 2019].

[32]Virtualbox.org. (n.d.). *Oracle VM VirtualBox*. [online] Available at: https://www.virtualbox.org/ [Accessed 24 Jan. 2019].

[33]Develapps. (n.d.). *RxSwift: Programación reactiva en Swift | Develapps*. [online] Available at: http://www.develapps.com/es/noticias/rxswift-programacion-reactiva-en-swift [Accessed 24 Jan. 2019].

[34]Firebase. (n.d.). *Configura una app cliente de Firebase Cloud Messaging en iOS | Firebase*. [online] Available at: https://firebase.google.com/docs/cloud-messaging/ios/client?authuser=1&hl=es-419 [Accessed 24 Jan. 2019].

[35]GitHub. (n.d.). *Alamofire/Alamofire*. [online] Available at: https://github.com/Alamofire/Alamofire [Accessed 24 Jan. 2019].

[36]GitHub. (n.d.). *tristanhimmelman/ObjectMapper*. [online] Available at: https://github.com/Hearst-DD/ObjectMapper [Accessed 24 Jan. 2019].

[37]YouTube. (n.d.). *Jared Davidson*. [online] Available at: https://www.youtube.com/user/Archetapp [Accessed 24 Jan. 2019].

[38]YouTube. (n.d.). *Lets Build That App*. [online] Available at: https://www.youtube.com/channel/UCuP2vJ6kRutQBfRmdcI92mA [Accessed 24 Jan. 2019].

[39]YouTube. (n.d.). *Sean Allen*. [online] Available at: https://www.youtube.com/channel/UCbTw29mcP12YlTt1EpUaVJw [Accessed 24 Jan. 2019].

[40]YouTube. (n.d.). *The Swift Guy*. [online] Available at: https://www.youtube.com/channel/UC-d1NWv5IWtIkfH47ux4dWA [Accessed 24 Jan. 2019].

[41]Davis, A. (1993). *Software requirements*. Englewood Cliffs, N.J.: PTR Prentice Hall.

[42]Larman, C. (2002). *Applying UML and patterns*. Upper Saddle River, N.J.: Prentice Hall PTR.

[43]Christensen, H. (2010). *Flexible, Reliable Software*. Hoboken: CRC Press.

[44]Cohn. (2005). *Agile Estimating and Planning*. Pearson India.

[45]Melton, J. and Simon, A. (2002). *SQL:1999*. San Francisco: Academic Press.